Thadomal Shahani Engineering College

Bandra (W.), Mumbai - 400 050.

CERTIFICATE

Certify that Mr./~~Miss~~ Diptanshu Mishra.

of I.T Department, Semester V with Roll No. 78 has completed a course of the necessary experiments in the subject Internet Programming under my supervision in the **Thadomal Shahani Engineering College** Laboratory in the year 2023 - 2024.

Teacher In- Charge — Head of the Department

Date 20/10/23. — Principal

CONTENTS

TSEC ENGINEERING COLLEGE

SR. NO.	EXPERIMENTS	PAGE NO.	DATE	TEACHERS SIGN.
1.	Develop a web app by using HTML tags. Elements, Attributes, Head, Body, Hyperlink, Images, audio, Frames Etc.	1	27-07	
2.	Using CSS and CSS3 enhance the web app developed in assignment 1. Color, Background, Font table, list CSS3, selectors, Pseudo class etc.	2	3/08/23	Signed 20/10/23
3.	Develop a webpage using the Bootstrap Framework, Grid system, Forms, Buttons, Navbar, Breadcrumbs etc.	3	3/08/23	
4.	a. WAP in JS to study conditional statements, loops and Function.	4	10/8/23	
	b. WAP on inheritance, Iterators etc.		10/8/23	Signed 20/10/23
5.	a) write a JS Program to study Arrow functions.	5	12/8/23	
	b) Write a JS Program.		12/8/23	
	c) Implement the concept of Promise.			
	d) Fetch (client server Communication).			
	e) Asynchronous Javascript.			
6)	a) WAP to implement Props and state.	6	24/8/23	Signed
	b) WAP to implement concept of forms.			
7)	a:- WAP to implement concept of React Hooks.	7	24/8/23	
	b) WAP to implement Concept of ReactJS Router and Animation.			

CONTENTS

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

Assignment No :1

Aim: Develop a Web Application by using HTML tags.

Theory: HTML is the standard markup language for creating Web pages. HTML stands for Hyper Text Markup Language

- HTML is the standard markup language for creating Web pages
- HTML describes the structure of a Web page
- HTML consists of a series of elements
- HTML elements tell the browser how to display the content
- HTML elements label pieces of content such as "this is a heading", "this is a paragraph", "this is a link", etc.

  Elements:

  The HTML **element** is everything from the start tag to the end tag:

  <tagname>Content goes here...</tagname>

  Examples of some HTML elements:

  <h1>My First Heading</h1>

  My first paragraph.

  The core tags of HTML are:

  1. <html>: Denotes the beginning and end of an HTML document.

  2. <head>: Houses meta-information about the page, such as the title and links to external resources.

  3. <body>: Encloses the visible content of the page, such as text, images, and multimedia.

  4. <h1> - <h6>: Define headings in descending order of importance.

  5. : Represents paragraphs of text.

  6. <a>: Creates hyperlinks to other web pages or resources.

  7. <img>: Embeds images into the page.

  8. and <ol>: Establish unordered and ordered lists, respectively.

  9. : Represents individual items within lists.

  10. and : Generic containers for grouping and styling elements.

  These tags form the building blocks for constructing the structure and content of web pages in HTML.

Assignment No :1

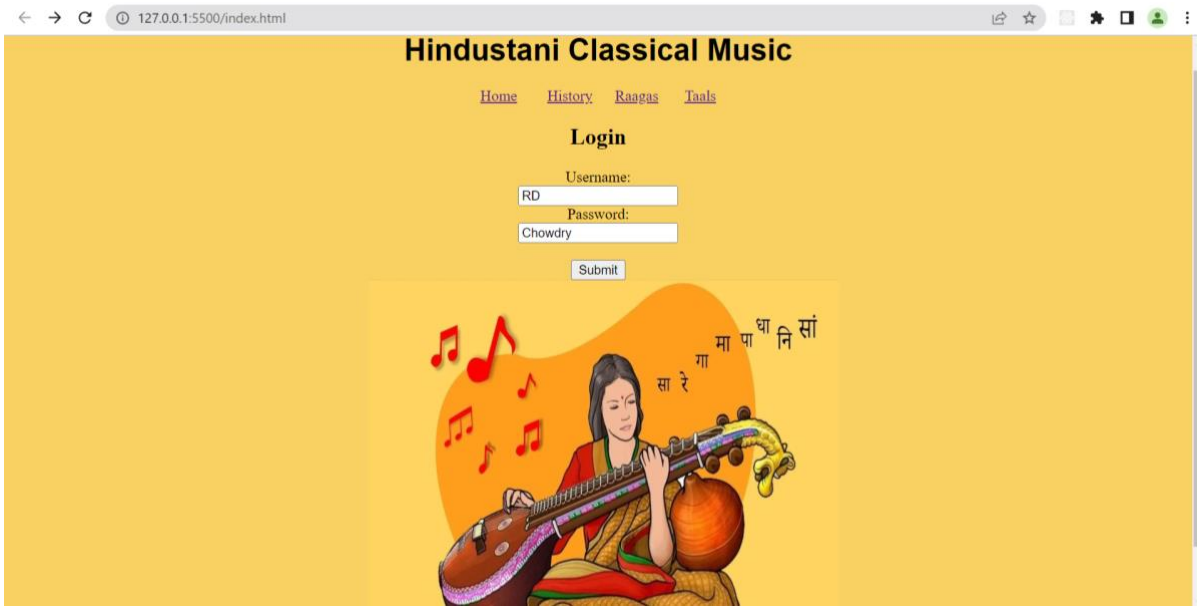
Implementation:

We Have Created a simple static website that consists of 5 pages. It is a Site called “Hindustani Classical Music and Its Components”. We used many elements and tags such as paragraph tag, heading tag, image, audio, video, iframe and tables etc.

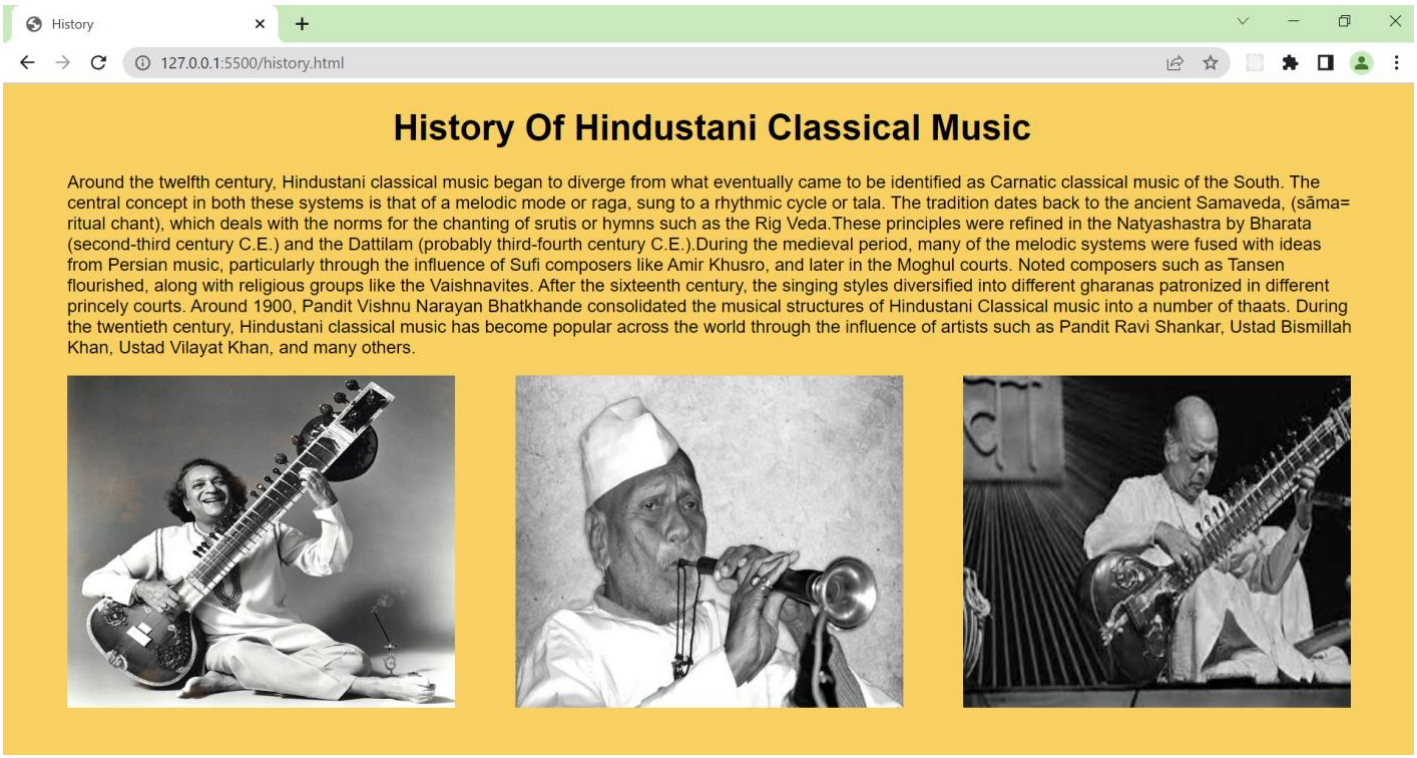
The Pages in our website are:

1. Home
2. History
3. Ragas
4. Taals
5. Contact Us.

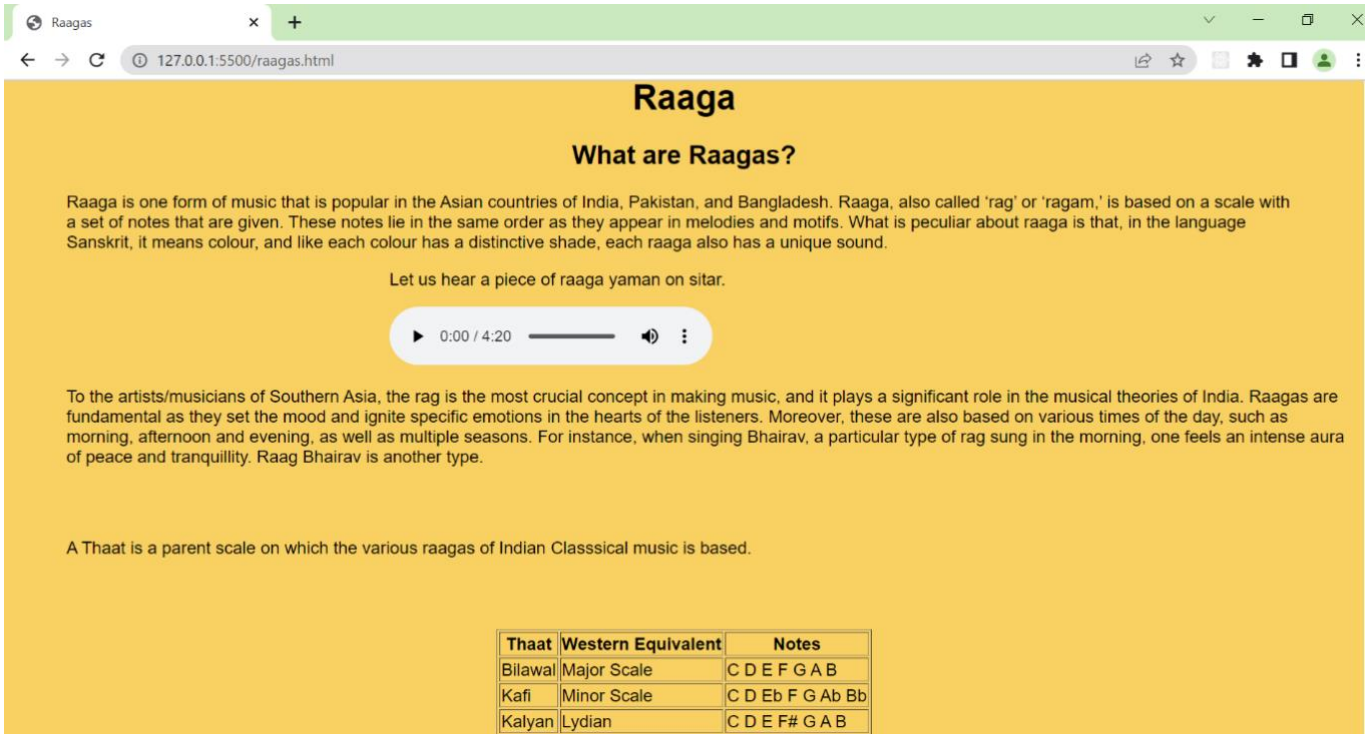
Listed Below are the screenshots of our website:

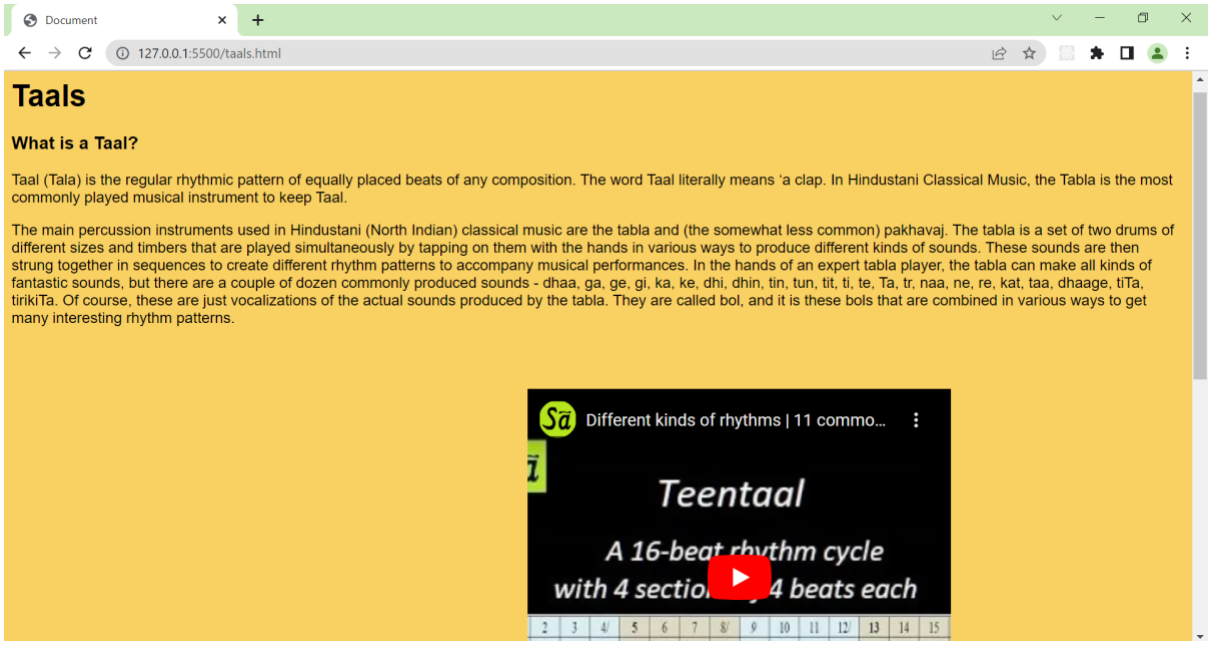


Assignment No :1

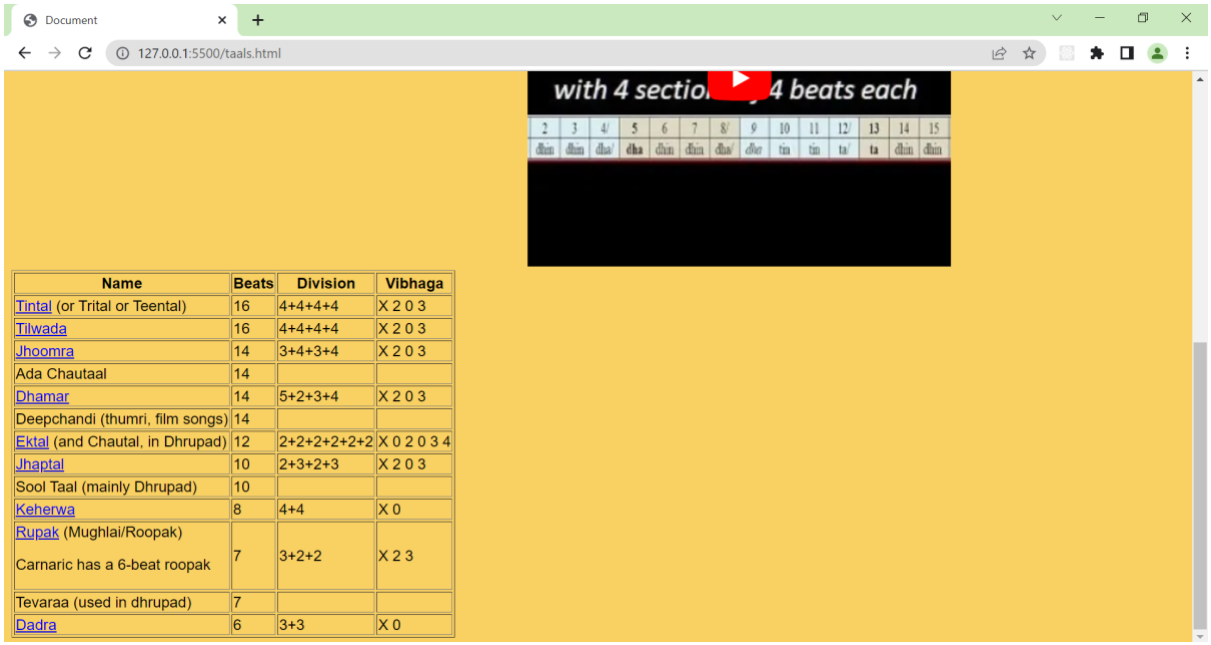


Assignment No :1





Assignment No :1



Name	Beats	Division	Vibhaga
Tintal (or Trital or Teental)	16	4+4+4+4	X 2 0 3
Tilwada	16	4+4+4+4	X 2 0 3
Jhoomra	14	3+4+3+4	X 2 0 3
Ada Chautaal	14		
Dhamar	14	5+2+3+4	X 2 0 3
Deepchandi (thumri, film songs)	14		
Ektal (and Chautal, in Dhrupad)	12	2+2+2+2+2+2	X 0 2 0 3 4
Jhaptal	10	2+3+2+3	X 2 0 3
Sool Taal (mainly Dhrupad)	10		
Keherwa	8	4+4	X 0
Rupak (Mughlai/Roopak) Carnaric has a 6-beat roopak	7	3+2+2	X 2 3
Tevaraa (used in dhrupad)	7		
Dadra	6	3+3	X 0

Conclusion:

We created a simple, static website only using vanilla HTML. Other important tools like CSS, Javascript were not used, hence the website looked bad and tacky. Therefore in our next assignment we will use CSS to style and beautify our website.

2.CSS

Aim: Using CSS and CSS3 enhance web application developed in Assignment 1.

Theory:

CSS (Cascading Style Sheets) is a fundamental technology used in web development to control the presentation and layout of HTML documents. It allows web developers to apply styles, such as colors, fonts, margins, and spacing, to HTML elements, thereby enhancing the overall visual appearance and user experience of a website. CSS operates on a "cascading" principle, where styles can be defined at different levels, and they cascade down to affect elements within the HTML document. This gives developers the flexibility to create consistent and visually appealing designs across multiple web pages.

CSS3 is the latest iteration of CSS, introducing a wide range of new features and enhancements. With CSS3, web developers gain access to more advanced techniques, such as animations, transitions, and transformations, which enable them to bring elements to life and create engaging user interactions. Additionally, CSS3 offers improved support for media queries, making websites responsive and adaptable to different screen sizes and devices. This crucial aspect of CSS3 ensures that users can enjoy a seamless experience, whether they are accessing the website on a desktop, tablet, or smartphone.

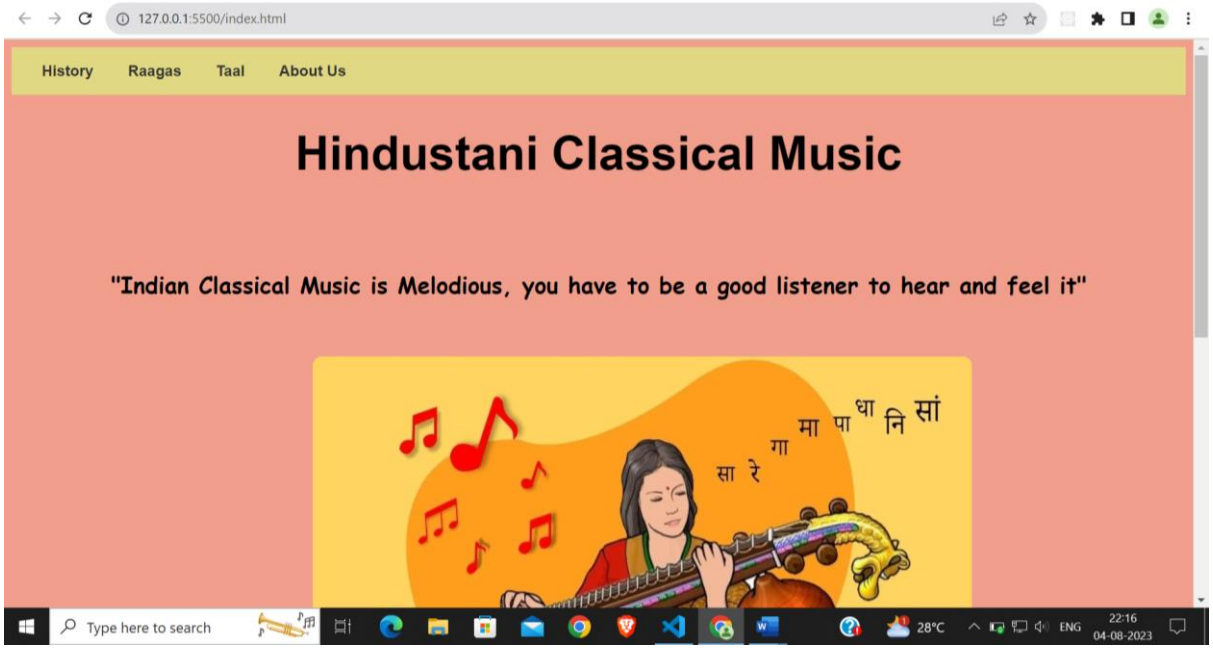
Here are some basic CSS properties that you can use for beautifying HTML elements:

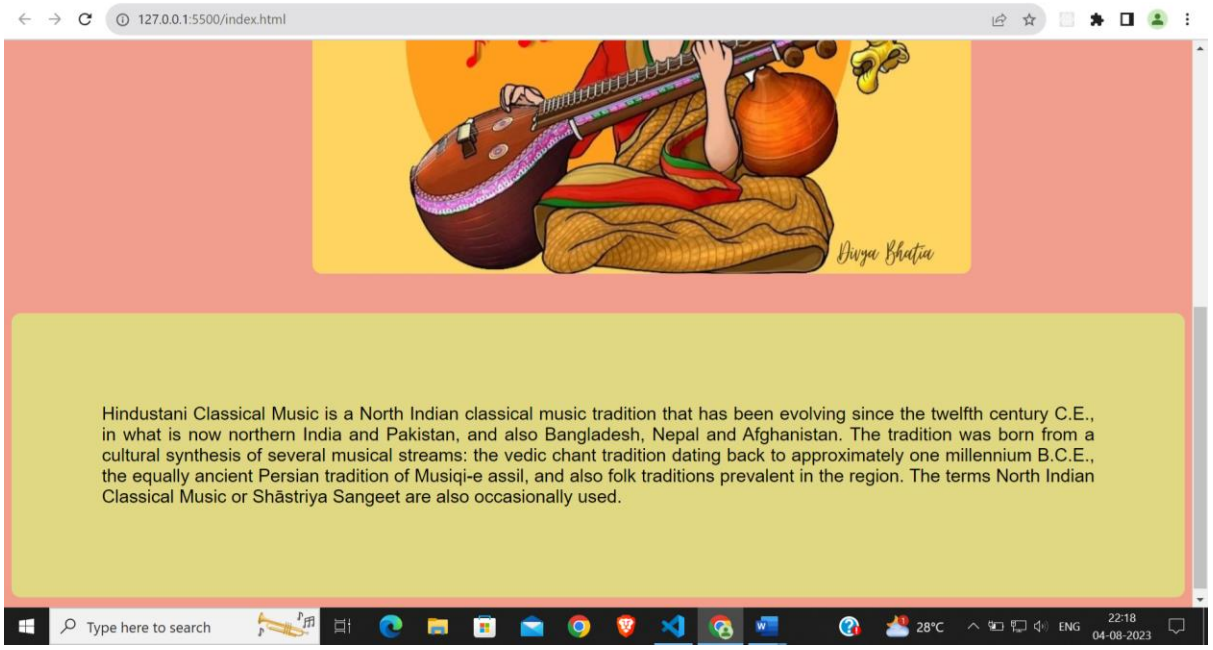
1. color: Sets the text color.
2. font-family: Specifies the font family to be used for text.
3. font-size: Defines the size of the font.
4. font-weight: Specifies the thickness of the font.
5. text-align: Aligns text content within an element (e.g., left, right, center).
6. background-color: Sets the background color of an element.
7. padding: Defines the space between the content and the border of an element.
8. margin: Sets the space outside an element's border.
9. border: Adds a border around an element.
10. border-radius: Rounds the corners of an element.
11. text-decoration: Applies decorations to text (e.g., underline, overline, line-through).
12. display: Defines how an element should be displayed (e.g., block, inline, inline-block).
13. width and height: Sets the width and height of an element.
14. position: Positions an element (e.g., relative, absolute, fixed).

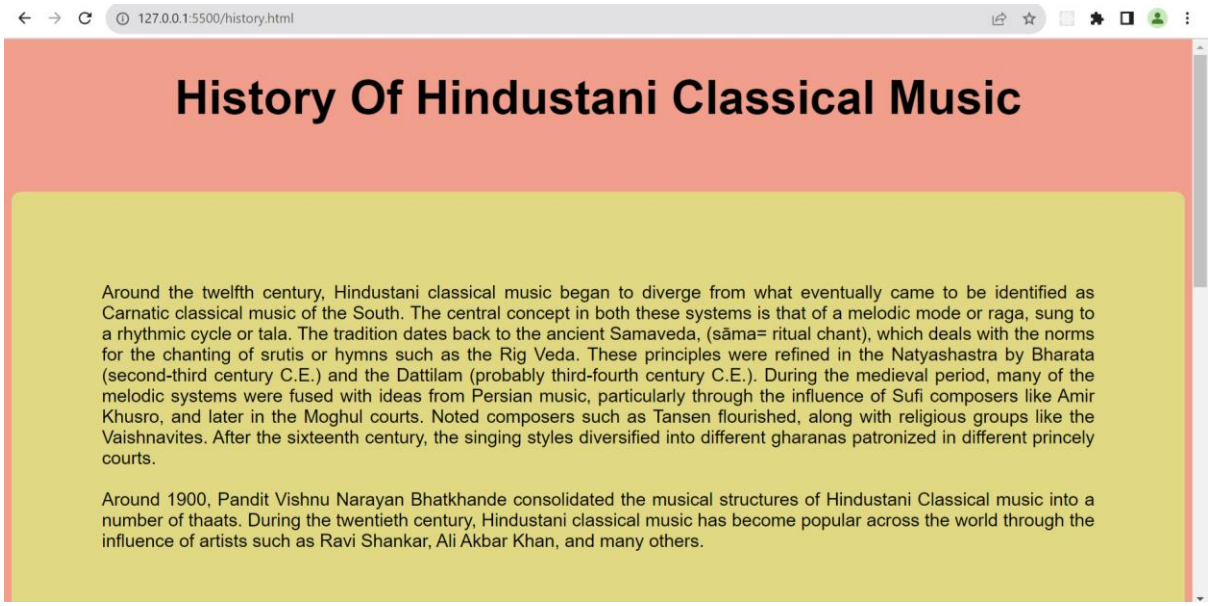
These CSS properties are essential for styling and beautifying HTML elements and can be used to create visually appealing and attractive web designs.

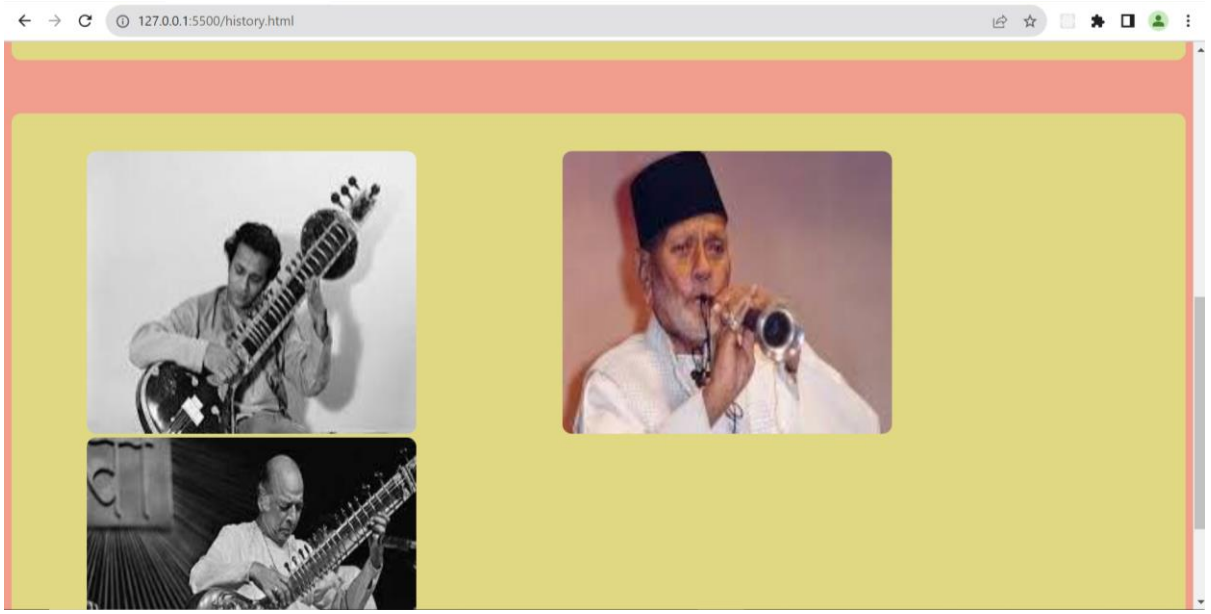
Implementation:

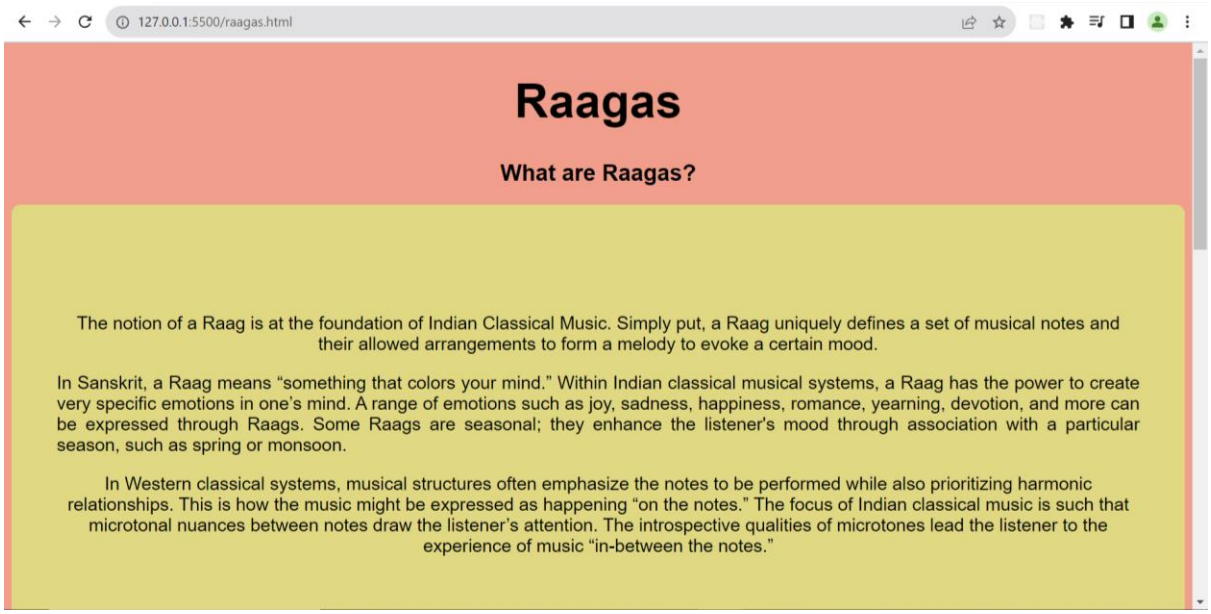
In the earlier assignment we had created an HTML assignment. Now, in assignment 2, we shall apply CSS to our site Hindustani Classical Music.





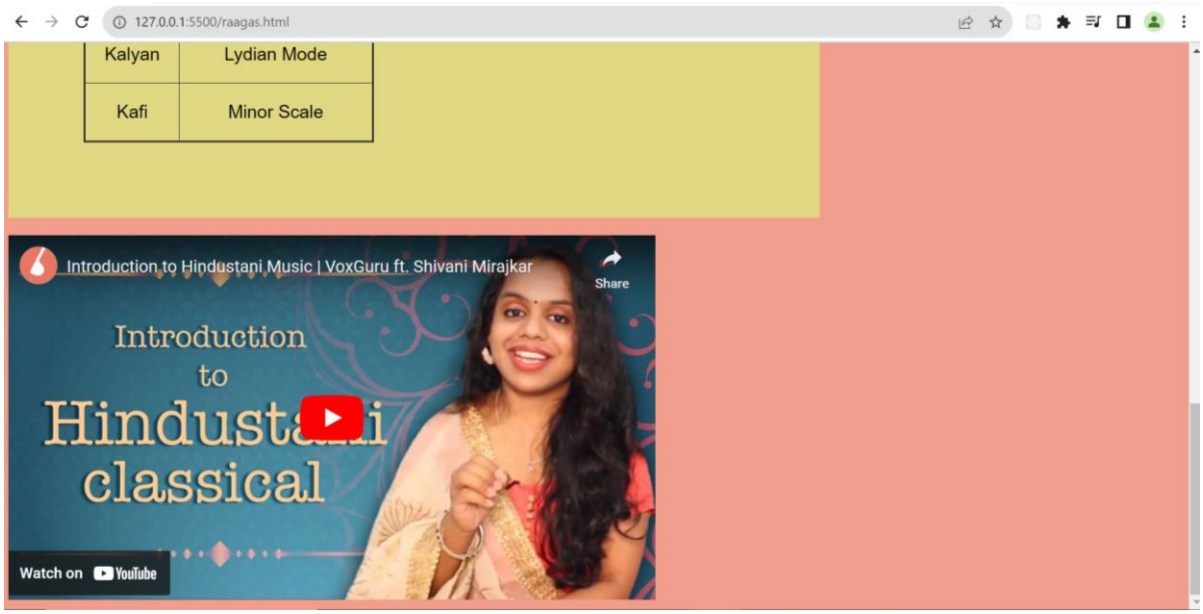

127.0.0.1:5500/history.html
History Of Hindustani Classical Music
Around the twelfth century, Hindustani classical music began to diverge from what eventually came to be identified as Carnatic classical music of the South. The central concept in both these systems is that of a melodic mode or raga, sung to a rhythmic cycle or tala. The tradition dates back to the ancient Samaveda, (sāma= ritual chant), which deals with the norms for the chanting of srutis or hymns such as the Rig Veda. These principles were refined in the Natyashastra by Bharata (second-third century C.E.) and the Dattilam (probably third-fourth century C.E.). During the medieval period, many of the melodic systems were fused with ideas from Persian music, particularly through the influence of Sufi composers like Amir Khusro, and later in the Moghul courts. Noted composers such as Tansen flourished, along with religious groups like the Vaishnavites. After the sixteenth century, the singing styles diversified into different gharanas patronized in different princely courts.
Around 1900, Pandit Vishnu Narayan Bhatkhande consolidated the musical structures of Hindustani Classical music into a number of thaats. During the twentieth century, Hindustani classical music has become popular across the world through the influence of artists such as Ravi Shankar, Ali Akbar Khan, and many others.

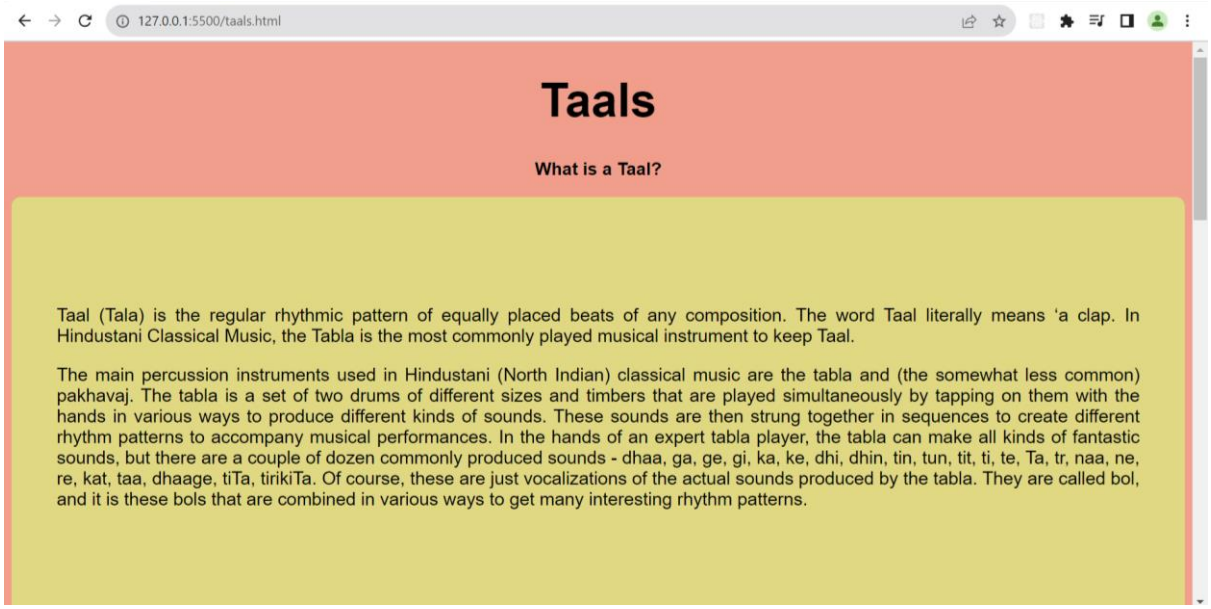

127.0.0.1:5500/history.html

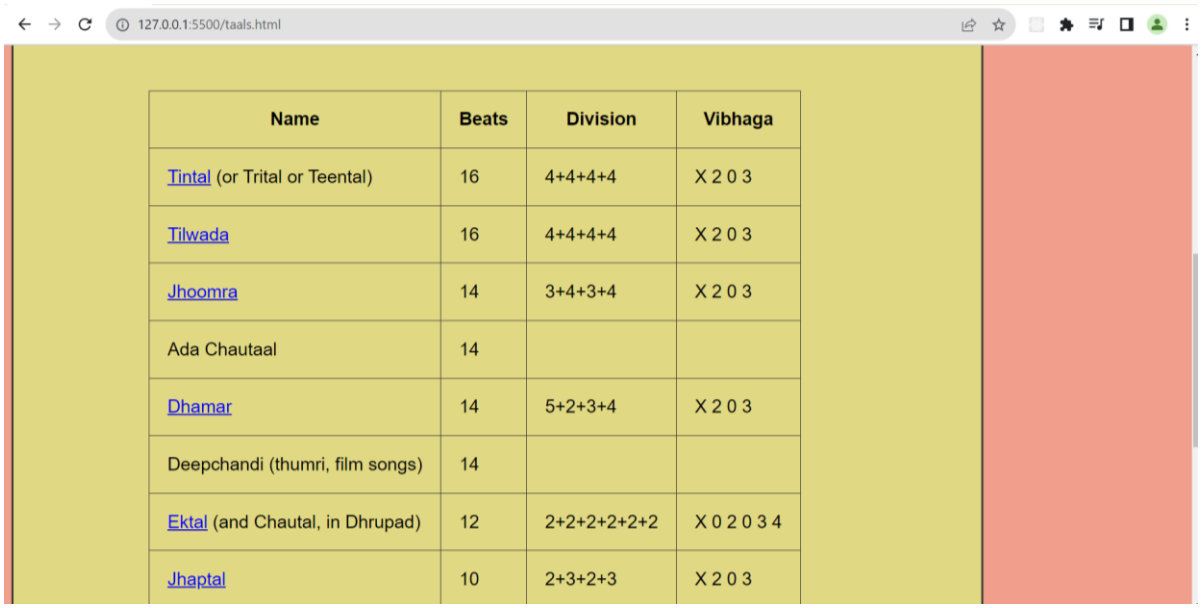

127.0.0.1:5500/raagas.html
Raagas
What are Raagas?
The notion of a Raag is at the foundation of Indian Classical Music. Simply put, a Raag uniquely defines a set of musical notes and their allowed arrangements to form a melody to evoke a certain mood.
In Sanskrit, a Raag means “something that colors your mind.” Within Indian classical musical systems, a Raag has the power to create very specific emotions in one’s mind. A range of emotions such as joy, sadness, happiness, romance, yearning, devotion, and more can be expressed through Raags. Some Raags are seasonal; they enhance the listener's mood through association with a particular season, such as spring or monsoon.
In Western classical systems, musical structures often emphasize the notes to be performed while also prioritizing harmonic relationships. This is how the music might be expressed as happening “on the notes.” The focus of Indian classical music is such that microtonal nuances between notes draw the listener’s attention. The introspective qualities of microtones lead the listener to the experience of music “in-between the notes.”


127.0.0.1:5500/raagas.html
Lets see various raagas and their Western Equivalents
Raaga
Western Equivalent
Bilawal
Major Scale
Kalyan
Lydian Mode
Kafi
Minor Scale

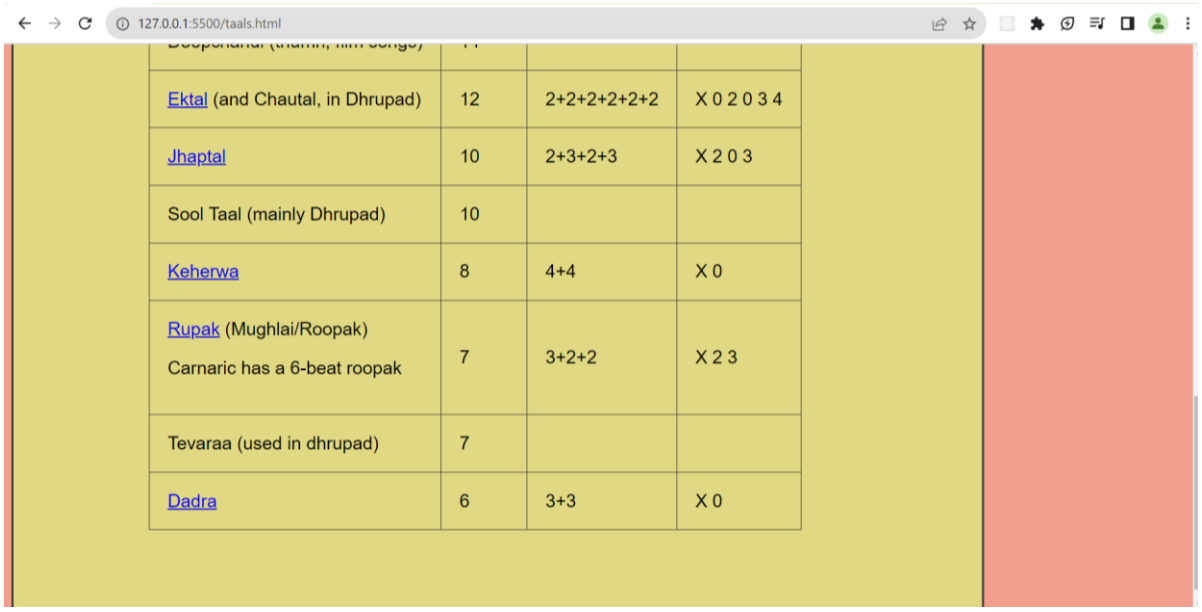
2.CSS


127.0.0.1:5500/raagas.html
Kalyan
Lydian Mode
Kafi
Minor Scale
Introduction to Hindustani Music | VoxGuru ft. Shivani Mirajkar
Share
Introduction
to
Hindustani
classical
Watch on YouTube


127.0.0.1:5500/taals.html
Taals
What is a Taal?
Taal (Tala) is the regular rhythmic pattern of equally placed beats of any composition. The word Taal literally means 'a clap. In Hindustani Classical Music, the Tabla is the most commonly played musical instrument to keep Taal.
The main percussion instruments used in Hindustani (North Indian) classical music are the tabla and (the somewhat less common) pakhavaj. The tabla is a set of two drums of different sizes and timbers that are played simultaneously by tapping on them with the hands in various ways to produce different kinds of sounds. These sounds are then strung together in sequences to create different rhythm patterns to accompany musical performances. In the hands of an expert tabla player, the tabla can make all kinds of fantastic sounds, but there are a couple of dozen commonly produced sounds - dhaa, ga, ge, gi, ka, ke, dhi, dhin, tin, tun, tit, ti, te, Ta, tr, naa, ne, re, kat, taa, dhaage, tiTa, tirikiTa. Of course, these are just vocalizations of the actual sounds produced by the tabla. They are called bol, and it is these bols that are combined in various ways to get many interesting rhythm patterns.



Name	Beats	Division	Vibhaga
Tintal (or Trital or Teental)	16	4+4+4+4	X 2 0 3
Tilwada	16	4+4+4+4	X 2 0 3
Jhoomra	14	3+4+3+4	X 2 0 3
Ada Chautaal	14		
Dhamar	14	5+2+3+4	X 2 0 3
Deepchandi (thumri, film songs)	14		
Ektal (and Chautal, in Dhrupad)	12	2+2+2+2+2+2	X 0 2 0 3 4
Jhaptal	10	2+3+2+3	X 2 0 3



Ektal (and Chautal, in Dhrupad)	12	2+2+2+2+2+2	X 0 2 0 3 4
Jhaptal	10	2+3+2+3	X 2 0 3
Sool Taal (mainly Dhrupad)	10		
Keherwa	8	4+4	X 0
Rupak (Mughlai/Roopak) Carnaric has a 6-beat roopak	7	3+2+2	X 2 3
Tevaraa (used in dhrupad)	7		
Dadra	6	3+3	X 0

Conclusion:

CSS (Cascading Style Sheets) played a pivotal role in beautifying our HTML in this assignment. By harnessing the power of CSS, we were able to transform plain, static HTML elements into visually captivating and aesthetically pleasing components. The use of CSS properties such as `color`, `font-family`, `font-size`, and `text-align` allowed us to style text and headings, making the content more engaging and readable.

In summary, CSS was the driving force behind the beautification of our HTML in this assignment. It empowered us to transform a simple collection of elements into a polished and visually appealing web page. As a result, understanding and leveraging CSS are crucial skills for web developers seeking to create captivating and well-designed web pages.

2.CSS

Assignment

Aim: Develop a web page using the Bootstrap framework. Grid system, Forms, Button, Navbar, Breadcrumb, Jumbotron should be used.

Code:

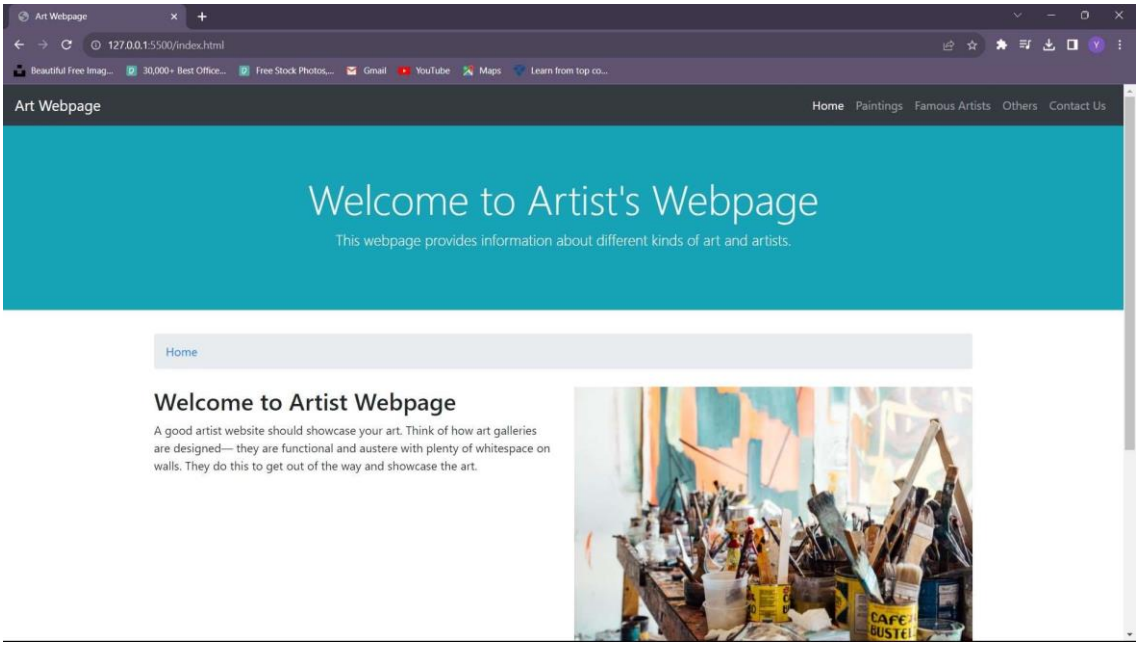
Index.html (landing page)

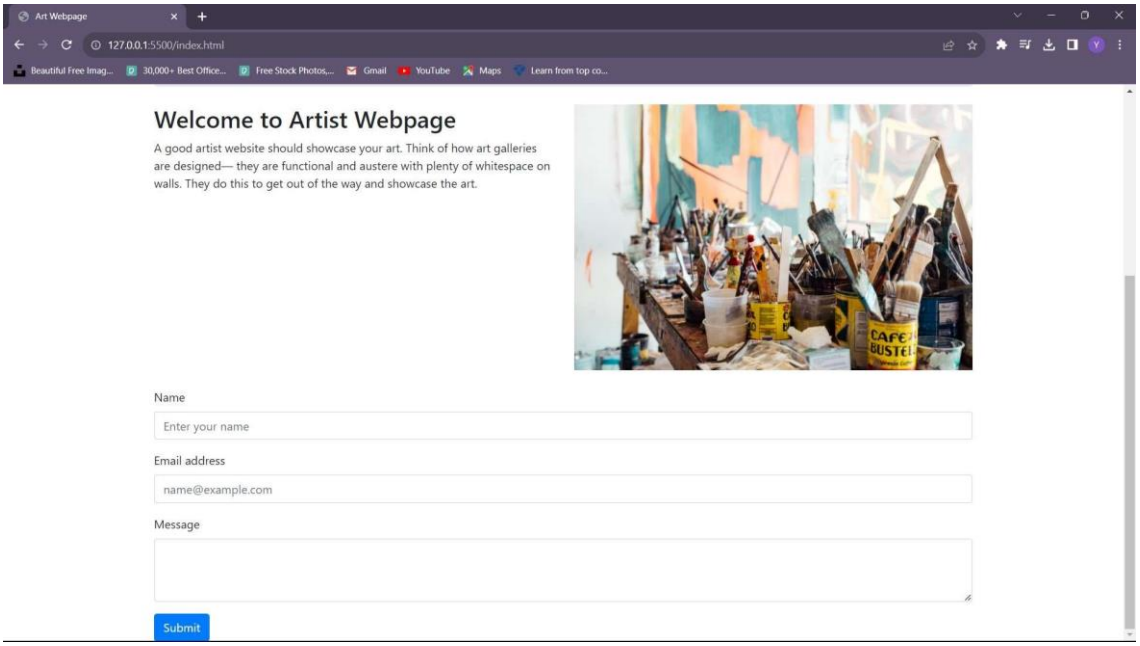
```
<!DOCTYPE html>
<html>
<head>
  <title>Art Webpage</title>
  <!--  Bootstrap CSS -->   <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.
min.css"> </head>
<body>
  <!-- Navbar -->
  <nav class="navbar navbar-expand-md navbar-dark bg-dark">
    <a class="navbar-brand" href="#">Art Webpage</a>     <button
class="navbar-toggler" type="button" datatoggle="collapse" data-
target="#navbarCollapse" aria-
controls="navbarCollapse" aria-expanded="false" aria-label="Toggle navigation">
      <span class="navbar-toggler-icon"></span>
    </button>
    <div class="collapse navbar-collapse" id="navbarCollapse">
      <ul class="navbar-nav ml-auto">
        <li class="nav-item active">
          <a class="nav-link" href="#">Home <span class="sronly">(current)</span></a>
```

```
      </li>
      <li class="nav-item">
        <a class="nav-link" href="paintings.html">Paintings</a>
      </li>
      <li class="nav-item">
        <a class="nav-link" href="famousartists.html">Famous
Artists</a>
      </li>
      <li class="nav-item">
        <a class="nav-link" href="#">Others</a>
      </li>
      <li class="nav-item">
        <a class="nav-link" href="#">Contact Us</a>
      </li>
    </ul>
  </div>
 </nav>

 <!-- Jumbotron -->
 <div class="jumbotron jumbotron-fluid text-center bg-info textwhite">
  <div class="container">
    <h1 class="display-4">Welcome to Artist's Webpage</h1>       <p
class="lead">This webpage provides information about different kinds of art and
artists.</p>

  </div>
 </div>
 <div class="container mt-4">
  <nav aria-label="breadcrumb">
    <ol class="breadcrumb">
      <li class="breadcrumb-item"><a href="index.html">Home</a></li>
    </ol>
  </nav>
 </div>

 <!-- Grid System -->
 <div class="container mt-4">
  <div class="row">
    <div class="col-md-6">
      <h2>Welcome to Artist Webpage</h2>
```

```
      <p>A good artist website should showcase your art. Think of how art galleries are designed— they are functional and austere with plenty of whitespace on walls. They do this to get out of the way and showcase the art. </p>
    </div>
    <div class="col-md-6">
      <img src="homepage.jpg" alt="Cricket Image" class="imgfluid">
   </div>
   </div>
  </div>

  <!-- Form -->
  <div class="container mt-4">     <form>
    <div class="form-group">
      <label for="name">Name</label>
      <input type="text" class="form-control" id="name"
placeholder="Enter your name">       </div>
    <div class="form-group">
      <label for="email">Email address</label>
      <input type="email" class="form-control" id="email"
placeholder="name@example.com">
    </div>
    <div class="form-group">
      <label for="message">Message</label>        <textarea
class="form-control" id="message" rows="3"></textarea>
    </div>
    <button type="submit" class="btn btn-primary">Submit</button>
   </form>
  </div>

  <!--  Bootstrap JS -->
  <script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.5.1/jquery.min.
js"></script>
```

```
 <script
src="https://maxcdn
.bootstrapcdn.com/
bootstrap/4.5.2/js/b
ootstrap.mi
n.js"></script>
</body>
</html>
```





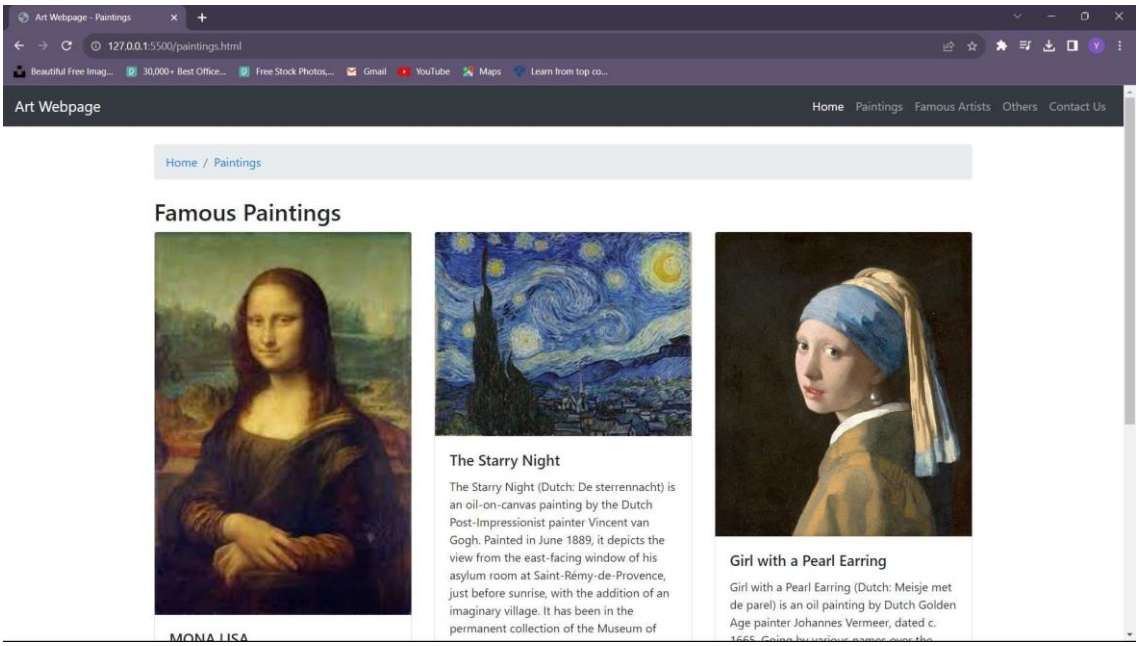
Pain ngs.html

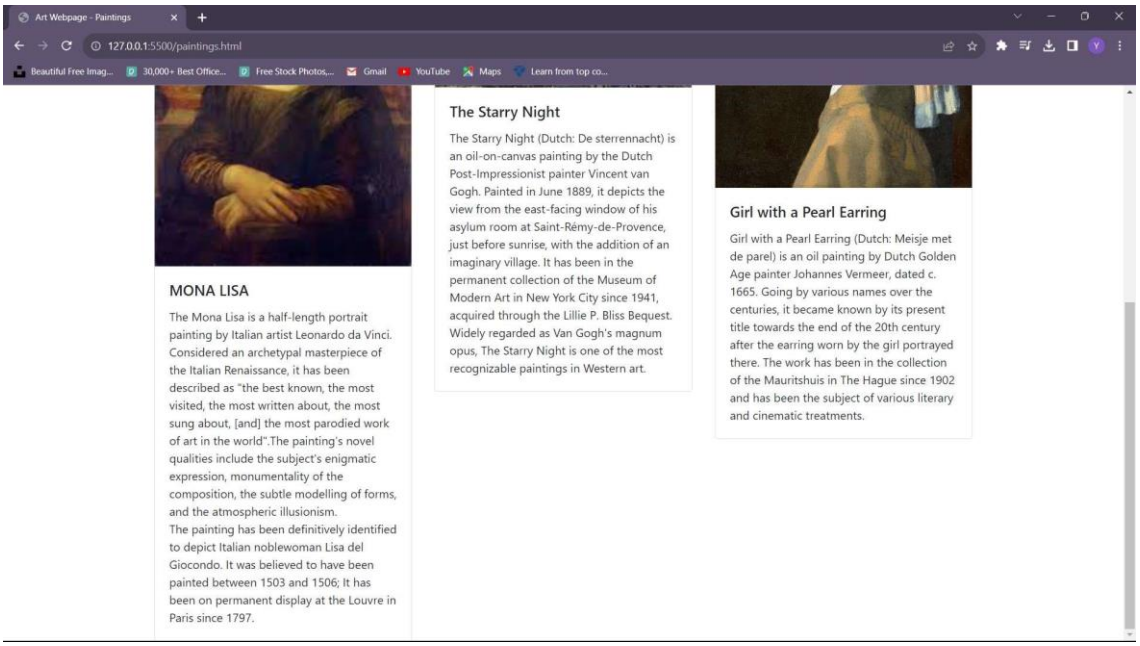
```
<!DOCTYPE html>
<html>
<head>
  <title>Art Webpage - Paintings</title>
  <!--  Bootstrap CSS -->   <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.
min.css"> </head>
<body>
  <!-- Navbar -->

  <nav class="navbar navbar-expand-md navbar-dark bg-dark">
    <a class="navbar-brand" href="#">Art Webpage</a>     <button
class="navbar-toggler" type="button" datatoggle="collapse" data-
target="#navbarCollapse" aria-
controls="navbarCollapse" aria-expanded="false" aria-label="Toggle navigation">
      <span class="navbar-toggler-icon"></span>
    </button>
    <div class="collapse navbar-collapse" id="navbarCollapse">
      <ul class="navbar-nav ml-auto">
        <li class="nav-item active">
          <a class="nav-link" href="index.html">Home </a>
        </li>
        <li class="nav-item">
          <a class="nav-link" href="paintings.html">Paintings <span class="sr-
only">(current)</span></a>
        </li>
        <li class="nav-item">
          <a class="nav-link" href="famousartists.html">Famous
Artists</a>
        </li>
        <li class="nav-item">
          <a class="nav-link" href="#">Others</a>
        </li>
        <li class="nav-item">
          <a class="nav-link" href="#">Contact Us</a>
        </li>
      </ul>
```

```
  </div>
 </nav>
 <!-- Breadcrumb -->
 <div class="container mt-4">
  <nav aria-label="breadcrumb">
   <ol class="breadcrumb">
    <li class="breadcrumb-item"><a
href="index.html">Home</a></li>        <li
class="breadcrumb-item"><a
href="paintings.html">Paintings</a></li>
   </ol>
  </nav>
 </div>

 <!-- Painting Content -->
  <div class="container mt-4">
   <h2>Famous Paintings</h2>
   <div class="row">
    <div class="col-md-4">
     <div class="card">
      <img src="paintings1.jpg"class="card-img-top" alt="Player 1">
      <div class="card-body">
       <h5 class="card-title">MONA LISA</h5>
       <p class="card-text">
        The Mona Lisa is a half-length portrait painting by Italian artist Leonardo da
Vinci.
        <br>
        Considered an archetypal masterpiece of the Italian Renaissance, it has been
described as "the best known, the most visited, the most written about, the most sung about,
[and] the most parodied work of art in the world".The painting's novel qualities include the
subject's enigmatic expression, monumentality of the composition, the subtle modelling of
forms, and the atmospheric illusionism.
        <br>
        The painting has been definitively identified to depict Italian noblewoman Lisa
del Giocondo. It was believed to have been painted between 1503 and 1506;  It has been on
permanent display at the Louvre in Paris since 1797.
       </p>
      </div>
```

```
          </div>
        </div>
        <div class="col-md-4">
          <div class="card">
            <img src="paintings2.jpg" class="card-img-top" alt="Player 2">
            <div class="card-body">
              <h5 class="card-title">The Starry Night</h5>                <p class="card-
text">The Starry Night (Dutch: De sterrennacht) is an oil-on-canvas painting by the Dutch
PostImpressionist painter Vincent van Gogh. Painted in June 1889, it depicts the view from
the east-facing window of his asylum room at Saint-Rémy-de-Provence, just before
sunrise, with the addition of an imaginary village. It has been in the permanent collection
of the Museum of Modern Art in New York City since 1941, acquired through the Lillie P.
Bliss Bequest. Widely regarded as Van Gogh's magnum opus, The Starry Night is one of
the most recognizable paintings in
Western art.</p>
            </div>
          </div>
        </div>
        <div class="col-md-4">
          <div class="card">
            <img src="paintings3.jpg" class="card-img-top" alt="Player 3">
            <div class="card-body">
              <h5 class="card-title"> Girl with a Pearl
Earring</h5>
              <p class="card-text">Girl with a Pearl Earring (Dutch: Meisje met de parel) is an
oil painting by Dutch Golden Age painter Johannes Vermeer, dated c. 1665. Going by
various names over the centuries, it became known by its present title towards the end of the
20th century after the earring worn by the girl portrayed there. The work has been in the
collection of the Mauritshuis in The Hague since 1902 and has been the subject of various
literary and cinematic treatments.</p>
            </div>
          </div>
        </div>
      </div>
    </div>
  </div>
```

```
  <!--  Bootstrap JS -->
  <script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.5.1/jquery.min.
js"></script>   <script
src="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.mi n.js"></script>
</body>
</html>
```



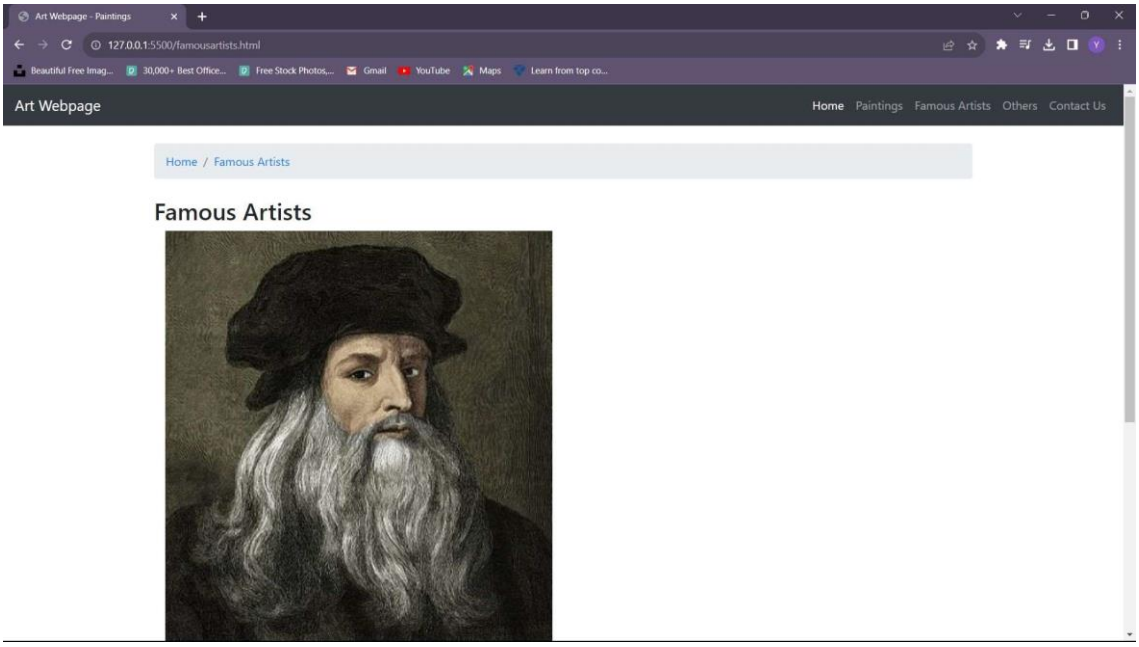


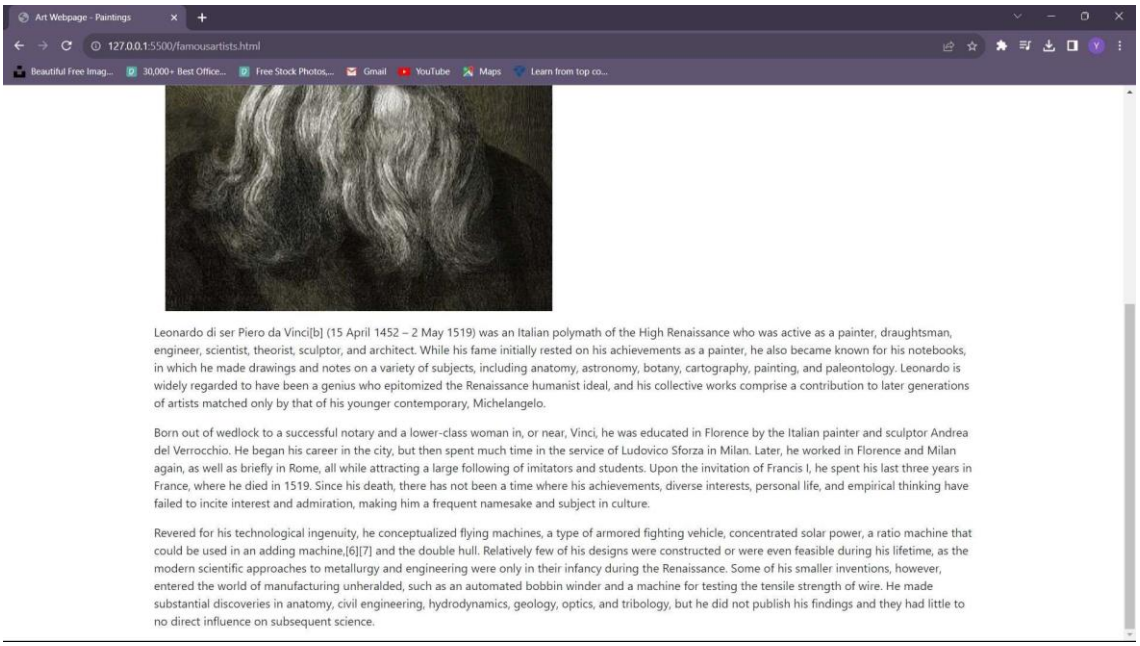
famousar sts.html

```
<!DOCTYPE html>
<html>
<head>
  <title>Art Webpage - Paintings</title>
  <!--  Bootstrap CSS -->   <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.
min.css">   <style>

  </style>
</head>
<body>
  <!-- Navbar -->

  <nav class="navbar navbar-expand-md navbar-dark bg-dark">
    <a class="navbar-brand" href="#">Art Webpage</a>     <button class="navbar-
toggler" type="button" datatoggle="collapse" data-target="#navbarCollapse"
ariacontrols="navbarCollapse" aria-expanded="false" aria-label="Toggle navigation">
      <span class="navbar-toggler-icon"></span>
    </button>
    <div class="collapse navbar-collapse" id="navbarCollapse">
      <ul class="navbar-nav ml-auto">
        <li class="nav-item active">
          <a class="nav-link" href="index.html">Home </a>
        </li>
        <li class="nav-item">
          <a class="nav-link" href="paintings.html">Paintings</a>
        </li>
        <li class="nav-item">
          <a class="nav-link" href="#">Famous Artists <span class="sr-
only">(current)</span></a>
        </li>
        <li class="nav-item">
          <a class="nav-link" href="#">Others</a>
        </li>
        <li class="nav-item">
```

```
        <a class="nav-link" href="#">Contact Us</a>
      </li>
    </ul>
  </div>
 </nav>

 <!-- Breadcrumb -->
 <div class="container mt-4">
  <nav aria-label="breadcrumb">
   <ol class="breadcrumb">
    <li class="breadcrumb-item"><a
href="index.html">Home</a></li>        <li
class="breadcrumb-item"><a
href="famousartists.html">Famous Artists</a></li>
   </ol>
  </nav>
 </div>
<div class="container mt-4">
 <h2>Famous Artists</h2>
 <div class="clearfix">
  <img src="FA1.jpg" class="col-md-6 float-md-end mb-3 ms-md-3" alt="Leonardo da
Vinci">
  <p>
   Leonardo di ser Piero da Vinci[b] (15 April 1452 – 2 May 1519) was an Italian
polymath of the High Renaissance who was active as a painter, draughtsman, engineer,
scientist, theorist, sculptor, and architect. While his fame initially rested on his
achievements as a painter, he also became known for his notebooks, in which he made
drawings and notes on a variety of subjects, including anatomy, astronomy, botany,
cartography, painting, and paleontology. Leonardo is widely regarded to have been a genius
who epitomized the Renaissance humanist ideal, and his collective works comprise a
contribution to later generations of artists matched only by that of his younger
contemporary, Michelangelo.
  </p>
  <p>
   Born out of wedlock to a successful notary and a lower-class woman in, or near, Vinci,
he was educated in Florence by the Italian painter and sculptor Andrea del Verrocchio. He
began his career in the city, but then spent much time in the service of Ludovico Sforza in
Milan. Later, he worked in Florence and Milan again, as well as briefly in Rome, all while
attracting a large following of imitators
```

```
and students. Upon the invitation of Francis I, he spent his last three years in France, where he died in 1519. Since his death, there has not been a time where his achievements, diverse interests, personal life, and empirical thinking have failed to incite interest and admiration, making him a frequent namesake and subject in culture.     </p>
   <p>
     Revered for his technological ingenuity, he conceptualized flying machines, a type of armored fighting vehicle, concentrated solar power, a ratio machine that could be used in an adding machine,[6][7] and the double hull. Relatively few of his designs were constructed or were even feasible during his lifetime, as the modern scientific approaches to metallurgy and engineering were only in their infancy during the Renaissance. Some of his smaller inventions, however, entered the world of manufacturing unheralded, such as an automated bobbin winder and a machine for testing the tensile strength of wire. He made substantial discoveries in anatomy, civil engineering, hydrodynamics, geology, optics, and tribology, but he did not publish his findings and they had little to no direct influence on subsequent science.
   </p>
 </div> </div>
```





Conclusion:

We implemented bootstrap by using its various features in multiple webpages like Grid system, Forms, Button, Navbar, Breadcrumb, Jumbotron.

Name: Diptanshu Mishra

Batch: T21

Roll no: 78

Aim: WAP in JS to study conditional Statements, Loops and Functions

Write a JavaScript code to change the background colour of the web page automatically at every 5 second.

Code:

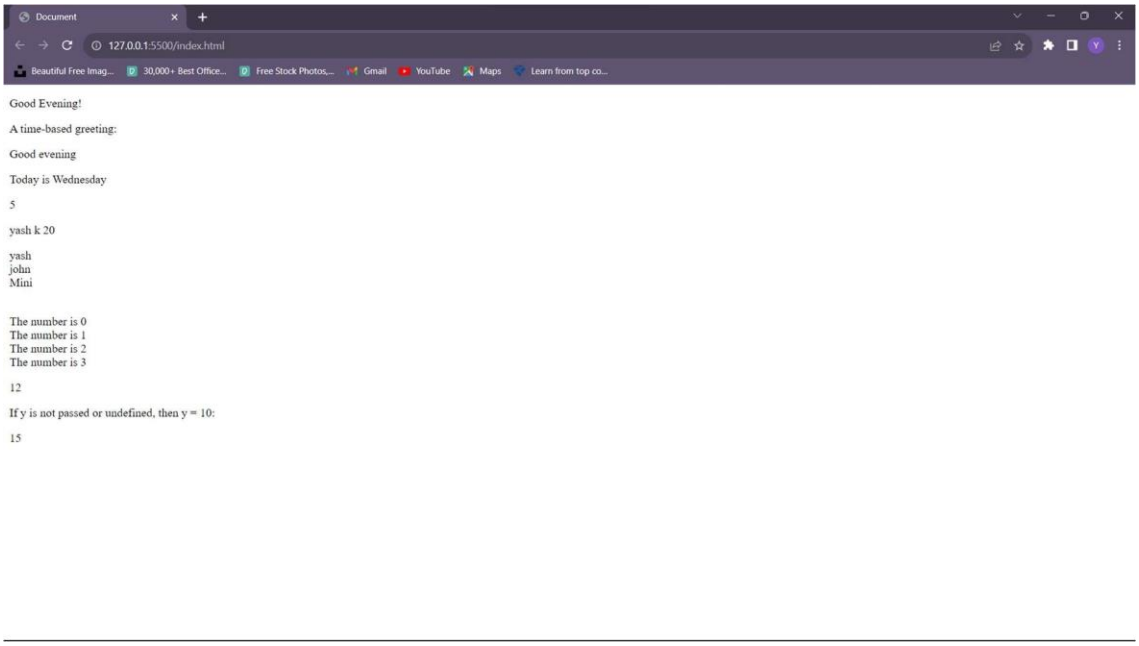
Index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initialscale=1.0">
    <title>Document</title>

</head>
<body>
    <p id="if">Good Evening!</p>
    <p>A time-based greeting:</p>
    <p id="ifelse"></p>
    <p id="switch"></p>
   <p id="for"></p>
   <p id="forin"></p>
   <p id="forof"></p>
   <p id="while"></p>
   <p id="regF"></p>
   <p>If y is not passed or undefined, then y = 10:</p>
<p id="parF"></p>
    <script src="node.js"></script>
</body>
</html>
```

Node.js

```
//if statement if (new Date().getHours() < 18) {
document.getElementById("if").innerHTML = "Good morning!"; }
//if else const hour = new Date().getHours();
let greeting; if (hour < 18) {
  greeting = "Good day";
} else {   greeting = "Good evening";
} document.getElementById("ifelse").innerHTML = greeting;
//switch switch (new Date().getDay()) {
case 0:       day = "Sunday";       break;
case 1:       day = "Monday";       break;
case 2:        day = "Tuesday";       break;
case 3:       day = "Wednesday";
break;     case 4:       day = "Thursday";
break;     case 5:       day = "Friday";
break;     case 6:
     day = "Saturday";
  }   document.getElementById("switch").innerHTML = "Today is " + day;
//for loop
```

```
let i = 5; for (let i = 0; i < 10; i++) {
} document.getElementById("for").innerHTML = i;
 //forin const person = {fname:"yash", lname:"k", age:20};  let txt
= ""; for (let x in person) {    txt += person[x] + " ";
}  document.getElementById("forin").innerHTML = txt;
 //forof
const name = ["yash", "john", "Mini"];
 let text = ""; for (let x of
name) {    text += x +
"<br>";
} document.getElementById("forof").innerHTML = text;
 //while let a = "" let j = 0; do {    a +=
"<br>The number is " + j;   j++; }
while (j < 4);
document.getElementById("while").innerHTML = a;

//regular function
const x = function (a, b) {return a * b}; document.getElementById("regF").innerHTML =
x(4, 3);

//parametrized function function
myFunction(x, y = 10) {
return x + y;
} document.getElementById("parF").innerHTML = myFunction(5);
```



Conclusion:

JavaScript's conditional statements, loops, and functions are integral components that empower developers to create dynamic and interactive web applications. By mastering these constructs, programmers can control program flow, efficiently manage repetitive tasks, and encapsulate reusable code. These foundational concepts form the basis of JavaScript programming and are essential for building sophisticated and responsive web experiences.

Assignment No: 4B

Aim: Write a Program on Inheritance, Iterators and Generators.

Theory:

Inheritance:

Inheritance is a fundamental concept in object-oriented programming that allows a new class (subclass or derived class) to inherit properties and methods from an existing class (superclass or base class). In JavaScript, inheritance can be achieved using prototype-based inheritance.

Prior to ES6 (ECMAScript 2015), JavaScript used prototype-based inheritance. Each object had a prototype, and properties and methods were looked up in the prototype chain if not found directly on the object. Constructors and prototypes were used to create and extend objects.

With ES6 and the introduction of the `class` syntax, JavaScript made inheritance more syntactically similar to other programming languages, but it's still based on prototypes under the hood.

Generators:

Generators are a special type of function in JavaScript that can be paused and resumed. They allow you to control the flow of execution, which can be useful for asynchronous programming, iterating over large datasets, and more. Generators use the `function*` syntax and the `yield` keyword to yield values from the generator function.

Generators are particularly useful for creating iterators (see the next point). They provide a way to produce a sequence of values on-demand, without needing to generate them all at once.

Iterators:

Iterators are a way to traverse a collection of data, such as an array or an object, one item at a time. Iterators provide a standard interface for looping over elements, and they are used in constructs like `for...of` loops.

An iterator object must implement a `next()` method that returns an object with two properties: `value` (the current value) and `done` (a boolean indicating if the iteration is complete).

JavaScript arrays, strings, maps, and sets are all iterable by default, which means you can use them with `for...of` loops.

Example Codes:

Assignment No: 4B

Inheritance:

```
class Animal {
  constructor(name) {
    this.name = name;
  }

  speak() {
    console.log(`${this.name} makes a sound.`);
  }
}
class Dog extends Animal {
  constructor(name, breed) {
    super(name);
    this.breed = breed;
  }
 speak() {
    console.log(`${this.name} barks.`);
  }
}
```

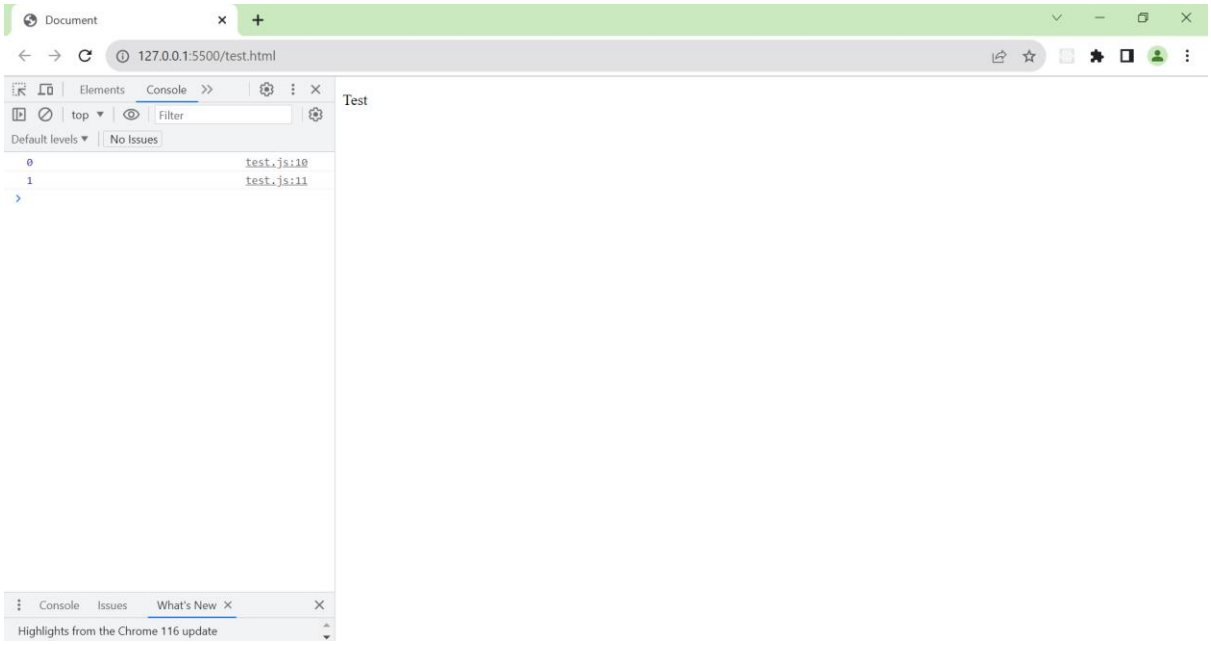
2.Generators:

```
function* countGenerator() {
  let count = 0;
  while (true) {
    yield count;
    count++;
  }
}

const counter = countGenerator();
```

Assignment No: 4B

```
console.log(counter.next().value);

console.log(counter.next().value);
```

Output:



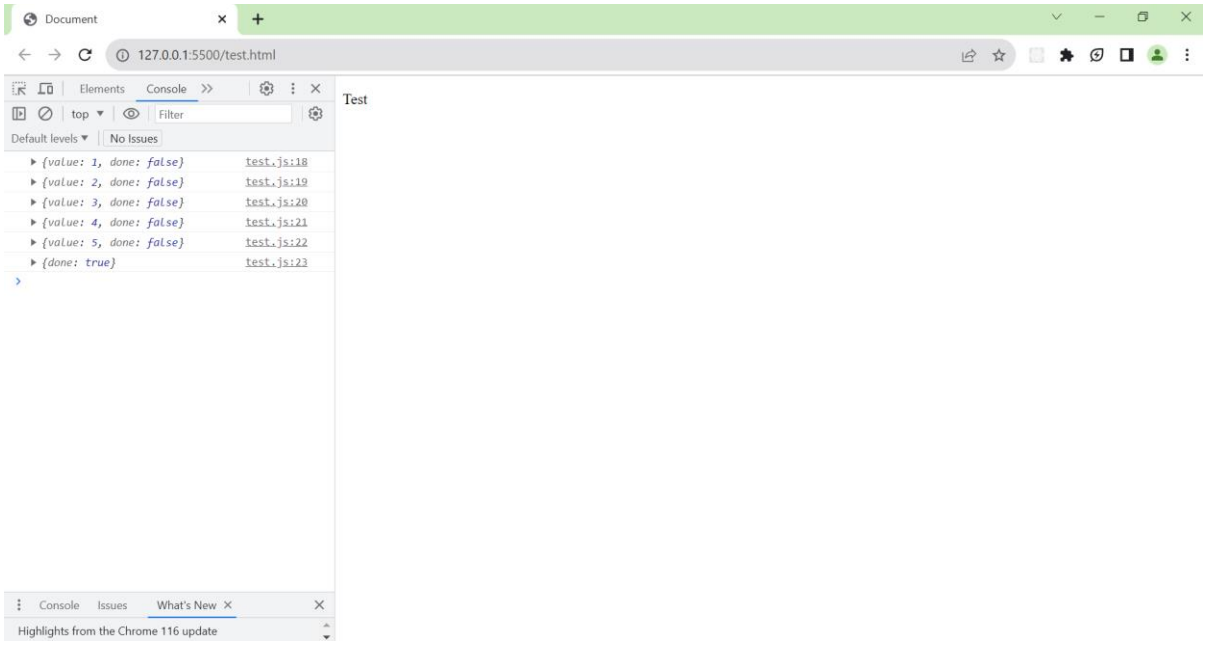
3. Iterators:

```
function createArrayIterator(array) {
  let index = 0;

  return {
    next: function() {
      if (index < array.length) {
        return { value: array[index++], done: false };
      } else {
        return { done: true };
      }
    }
  };
}
```

Assignment No: 4B

```
const numbers = [1, 2, 3, 4, 5];

const iterator = createArrayIterator(numbers);


console.log(iterator.next());

console.log(iterator.next());

console.log(iterator.next());

console.log(iterator.next());

console.log(iterator.next());

console.log(iterator.next());
```

Output:



Conclusion:

In conclusion we can say that, inheritance is about creating relationships between classes, generators allow for controlled asynchronous programming and custom iteration, and iterators provide a standardized way to loop over collections of data.

Assignment 5A

Aim: Write a JavaScript Program to study DOM Manipulation and CSS Manipulations.

Theory:

DOM manipulation refers to the process of using programming languages like JavaScript to interact with and modify the Document Object Model (DOM) of a web page. The DOM is a programming interface provided by web browsers that represents the structured content of a webpage, such as elements, attributes, and their relationships. It essentially provides a way for scripts to access, modify, and update the content and structure of a web page dynamically.

DOM manipulation is crucial for creating dynamic and interactive web applications. Through DOM manipulation, you can achieve various tasks, such as:

1. Changing Content: You can modify the text, attributes, and properties of HTML elements. For example, updating the text of a paragraph, changing the source of an image, or modifying the value of an input field.

2. Adding and Removing Elements: You can dynamically add new elements to the DOM or remove existing elements. This is often used for adding new content, forms, or dynamically generated components.

3. Event Handling: You can attach event listeners to elements to respond to user interactions like clicks, mouse movements, keypresses, etc.

4. Styling: You can change the CSS styles of elements, enabling you to create animations, toggle visibility, and more.

5. Fetching and Sending Data: You can use DOM manipulation to fetch data from a server (using APIs like Fetch or XMLHttpRequest() and update the page with the received data.

6. Creating Interactive Interfaces: You can build interactive features like dropdown menus, sliders, accordions, and modal dialogs.

Overall, DOM manipulation is a fundamental concept in web development that empowers developers to create dynamic, responsive, and interactive web applications.

Code:

index.html

Assignment 5A

```
<!DOCTYPE html>
<html>
<head>
  <title>DOM and CSS Manipulation Example</title>
  <style>
    #myParagraph {
      padding: 20px;
      background-color: lightgray;
      transition: background-color 0.3s ease;
    }

    #changeColorButton {
      margin-top: 10px;
    }
  </style>
</head>
<body>
  <p id="myParagraph">Click the button to change the color!</p>
  <button id="changeColorButton">Change Color</button>
  <script src="script.js"></script>
</body>
</html>
```

Index.js

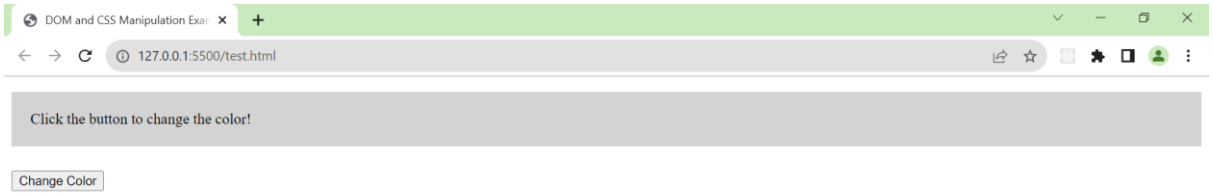
```
// script.js
document.addEventListener('DOMContentLoaded', function() {
  const paragraph = document.getElementById('myParagraph');
```

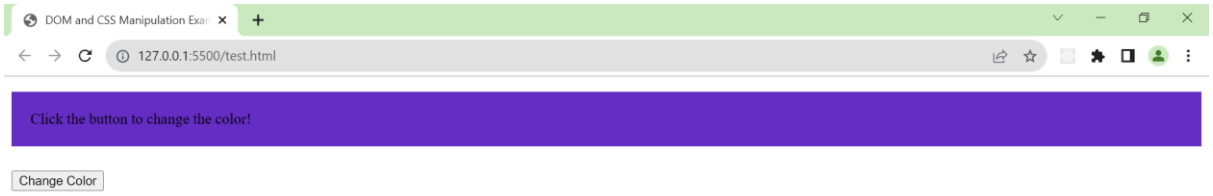
Assignment 5A

```
    const button = document.getElementById('changeColorButton');

    button.addEventListener('click', function() {
        // Generate a random color
        const randomColor = getRandomColor();

        // Apply the new color to the paragraph's background
        paragraph.style.backgroundColor = randomColor;
    });

    function getRandomColor() {
        const letters = '0123456789ABCDEF';
        let color = '#';
        for (let i = 0; i < 6; i++) {
            color += letters[Math.floor(Math.random() * 16)];
        }
        return color;
    }
});
```

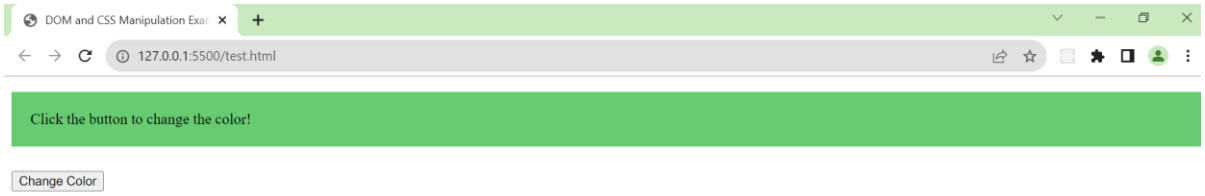
Output:

Assignment 5A





Assignment 5A



Conclusion:

In this Assignment, we learnt that DOM manipulation is crucial for creating dynamic and interactive web applications and it is a programming interface provided by web browsers that represents the structured content of a webpage, such as elements, attributes, and their relationships. It essentially provides a way for scripts to access, modify, and update the content and structure of a web page dynamically.

Assignment 5B

Aim: Write a JavaScript program to:

a. Implement the concept of Promise(call-back)
b. Fetch (Client Server Communication)

Theory:

In JavaScript, a Promise is an object that represents a value that might be available now, or in the future, or never. It's a way to handle asynchronous operations in a more organized and manageable manner, allowing you to write code that deals with asynchronous tasks without falling into deeply nested callback hell.

Promises have three states:

1. Pending: The initial state, when the Promise is created and its asynchronous operation is ongoing.

2. Fulfilled: The state when the asynchronous operation has completed successfully, and the Promise has a resulting value.

3. Rejected: The state when the asynchronous operation encountered an error or was unsuccessful, and the Promise has a reason for failure.

Here's how you can create a Promise:

```
const myPromise = new Promise((resolve, reject) => {
  // Perform some asynchronous operation
  const result = doSomethingAsync();

  if (result) {
    resolve(result); // Fulfill the promise with a value
  } else {
    reject("An error occurred"); // Reject the promise with a reason
  }
});
```

Assignment 5B

You can then use `.then()` and `.catch()` methods to handle the fulfilled and rejected states respectively:

```
myPromise
  .then(result => {
    console.log("Fulfilled:", result);
  })
  .catch(error => {
    console.error("Rejected:", error);
  });
```

In addition to this, you can also use the `.finally()` method, which is called regardless of whether the Promise is fulfilled or rejected. Promises also support chaining using the `.then()` method, which can be useful for sequential asynchronous operations.

ES6 also introduced the async/await syntax, which provides a more synchronous-looking way to work with Promises. Here's an example:

```
async function fetchData() {
  try {
    const result = await myPromise;
    console.log("Fulfilled:", result);
  } catch (error) {
    console.error("Rejected:", error);
  }
}
```

This allows you to write asynchronous code that appears more similar to synchronous code, making it easier to read and understand.

Assignment 5B

Promises are a fundamental concept in modern JavaScript development, making it easier to manage and reason about asynchronous operations.

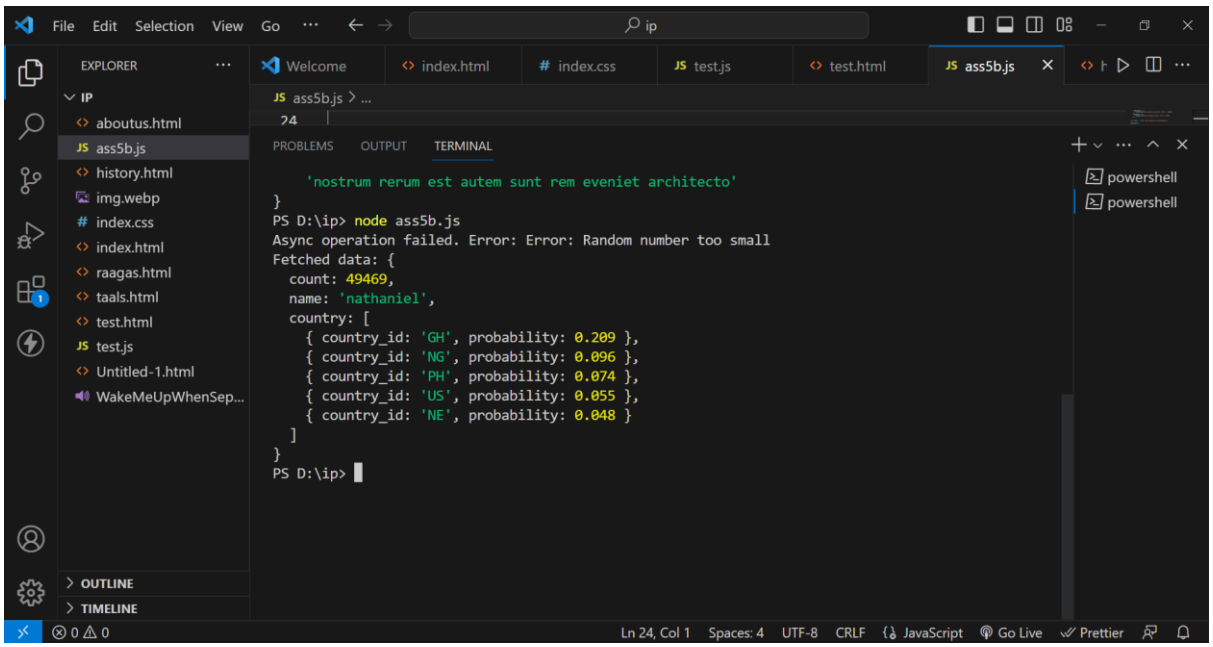
Code:

```
// Part A: Implementing Promise


function simulateAsyncOperation() {
    return new Promise((resolve, reject) => {
        setTimeout(() => {
            const randomNumber = Math.random();
            if (randomNumber > 0.5) {
                resolve(randomNumber);
            } else {
                reject("Error: Random number too small");
            }
        }, 1000);
    });
}


simulateAsyncOperation()
    .then(result => {
        console.log("Async operation successful. Result:", result);
    })
    .catch(error => {
        console.error("Async operation failed. Error:", error);
    });



// Part B: Using fetch for Client-Server Communication
```

Assignment 5B

```
const apiUrl = 'https://jsonplaceholder.typicode.com/posts/1';


fetch(apiUrl)
  .then(response => {
    if (!response.ok) {
      throw new Error(`HTTP error! Status: ${response.status}`);
    }
    return response.json();
  })
  .then(data => {
    console.log("Fetched data:", data);
  })
  .catch(error => {
    console.error("Fetch error:", error);
  });
```

Output:



Assignment 5B

Conclusion:

In this assignment, we learnt that Promises are a fundamental concept in modern JavaScript development, making it easier to manage and reason about asynchronous operations and In addition to this, you can also use the `.finally()` method, which is called regardless of whether the Promise is fulfilled or rejected. Promises also support chaining using the `.then()` method, which can be useful for sequential asynchronous operations.

Assignment: 6A

Aim: Write a program to implement the concept of Props and State.

Theory:

In React, "props" is a shorthand for "properties," and it refers to a mechanism for passing data from a parent component to a child component. Props are a way to provide input or configuration to a component and make it dynamic and reusable. They allow you to customize and control the behavior and appearance of a component without modifying its internal logic.

Here's a basic overview of how props work in React:

1. Passing Props from Parent to Child:

In React, you create components that can be reused throughout your application. When you use a component within another component's JSX code, you can provide it with attributes, also known as props. These props are passed down from the parent component to the child component.

2. Accessing Props in Child Component:

In the child component, the props are accessible as an object parameter. You can access the data passed from the parent component through this object.

Props are read-only, which means that a child component cannot modify the props it receives from its parent. If the child component needs to manage and update its own state, it should use React's internal state management with the `useState` hook or class component state.

Props provide a way to create reusable components that can be easily customized and used throughout your application. They promote the idea of component composition and help to keep components isolated and focused on their specific responsibilities.

States:

In React, "state" refers to an object that holds dynamic data that can change over time and affects the behavior and appearance of a component. Unlike props, which are passed from a parent component and are read-only within the child component, state is managed internally by the component itself. When the state of a component changes, React re-renders the component to reflect those changes in the user interface.

Assignment: 6A

Here's a basic overview of how state works in React:

1. Initializing State:

You can initialize the state of a component using the `useState` hook (for functional components) or the `this.state` property (for class components). The `useState` hook returns an array with two elements: the current state value and a function to update that value.

```
import React, { useState } from 'react';
function Counter() {
  const [count, setCount] = useState(0);

 const increment = () => {
   setCount(count + 1);
  };

  return (
   <div>
    <p>Count: {count}</p>
    <button onClick={increment}>Increment</button>
   </div>
  );
}
```

2. Updating State:

To update the state, you should never directly modify the state object. Instead, you use the state update function provided by `useState` or `this.setState()` method (for class components). When the state is updated using these functions, React will automatically trigger a re-render of the component with the new state values.

Assignment: 6A

3. Class Components and `this.state`:

In class components, the state is managed using the `this.state` object. When you update the state using `this.setState()`, React will merge your updates into the existing state.

```
import React, { Component } from 'react';

class Counter extends Component {
  constructor(props) {
    super(props);
    this.state = { count: 0 };
  }

  increment = () => {
    this.setState({ count: this.state.count + 1 });
  };

  render() {
    return (
      <div>
        <p>Count: {this.state.count}</p>
        <button onClick={this.increment}>Increment</button>
      </div>
    );
}
  }
```

Using state, you can create interactive and dynamic user interfaces in React. State management becomes especially important when components need to respond to user interactions, API responses, and other dynamic changes. However, it's important to manage state carefully to avoid unnecessary re-renders and potential bugs caused by stale data.

Assignment: 6A

Code 1:

```
document.addEventListener('DOMContentLoaded', function() {
    const paragraph = document.getElementById('myParagraph');
    const button = document.getElementById('changeColorButton');

    button.addEventListener('click', function() {

        const randomColor = getRandomColor();


        paragraph.style.backgroundColor = randomColor;
    });

    function getRandomColor() {
        const letters = '0123456789ABCDEF';
        let color = '#';
        for (let i = 0; i < 6; i++) {
            color += letters[Math.floor(Math.random() * 16)];
        }
        return color;
    }
});
```

Code 2:

```
import React from 'react';

function DateTimeDisplay(props) {
  return (
    <div>
      <p>Current Date: {props.currentDate.toLocaleDateString()}</p>
      <p>Current Time: {props.currentDate.toLocaleTimeString()}</p>
    </div>
  );
}

function App() {
  const currentDate = new Date();

  return (
    <div>
      <h1>Date and Time Display</h1>
      <DateTimeDisplay currentDate={currentDate} />
    </div>
  );
}
```

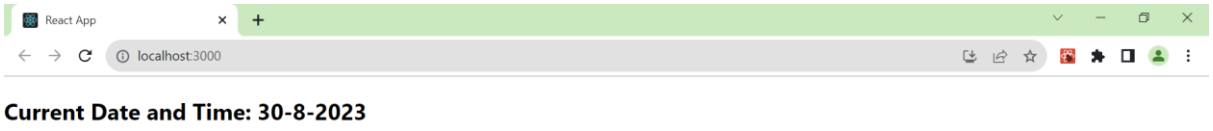
Assignment: 6A

```
export default App;
```

Output:

Output:

Assignment: 6A



Conclusion:

In this assignment we learnt that "props" is a shorthand for "properties," and it refers to a mechanism for passing data from a parent component to a child component. Props are a way to provide input or configuration to a component and make it dynamic and reusable and ‘State’ refers to an object that holds dynamic data that can change over time and affects the behavior and appearance of a component. Unlike props, which are passed from a parent component and are read-only within the child component, state is managed internally by the component itself.

Assignment No: 6B

Aim: Write a ReactJS code to implement the concept of forms and events.

Theory:

In React, an event is a user interaction or system-generated action that can trigger specific behavior in a React component. Events in React are similar to events in traditional HTML and JavaScript but are handled differently due to React's virtual DOM and component-based architecture. React components can respond to events such as clicks, input changes, key presses, and more.

Here are a couple of common types of events and their usage in React:

1. Click Event:

- This event occurs when a user clicks an element, such as a button or a link.

- You can handle click events in React using the onClick attribute on elements like buttons or links. For example:

<button onClick={handleClick}>Click me</button>

2. Change Event:

- Change events are commonly used with form elements like textboxes and checkboxes.

- You can handle change events using the onChange attribute. For example:

<input type="text" onChange={handleChange} />

Now, let's discuss textbox and textarea in React:

- Textbox (Input):

- In React, you can create a textbox (input field) using the <input> element.

- To capture user input from a textbox, you typically use the value and onChange attributes.

Assignment No: 6B

<input type="text" value={this.state.textValue} onChange={this.handleTextChange} />

In this example, this.state.textValue is used to control the value of the textbox, and this.handleTextChange is the event handler for changes in the textbox.

- Textarea:

- A textarea in React is created using the <textarea> element.

- Similar to a textbox, you can control the content of a textarea using the value and onChange attributes.

<textarea value={this.state.textareaValue} onChange={this.handleTextareaChange}></textarea>

In this case, this.state.textareaValue stores the content of the textarea, and this.handleTextareaChange is the event handler for changes in the textarea.

You'll need to define the corresponding event handlers (handleTextChange and handleTextareaChange in this case) in your component to handle user input and update the component's state accordingly.

Remember that in React, the component's state should be used to manage and reflect the current value of form elements like textboxes and textareas, and event handlers help update this state when user interactions occur.

Code:

RegistrationForm.js

```
import React, { Component } from 'react';
```

Assignment No: 6B

```
import './RegistrationForm.css';

class RegistrationForm extends Component {
  constructor(props) {
    super(props);
    this.state = {
      name: '',
      email: '',
      gender: 'male',
      message: '',
      subscribe: false,
    };
  }

  handleSubmit = (e) => {
    e.preventDefault();
    console.log('Form submitted with data:', this.state);
  };

  handleInputChange = (e) => {
    const { name, value, type, checked } = e.target;
    const newValue = type === 'checkbox' ? checked : value;
    this.setState({ [name]: newValue });
  };

  handleReset = () => {
    this.setState({
      name: '',
      email: '',
      gender: 'male',
      message: '',
      subscribe: false,
    });
  };

  handleKeyDown = (e) => {
    if (e.key === 'Enter') {
      this.handleSubmit(e);
    }
  };

  render() {
    return (
      <div className="registration-form">
        <h2>Registration Form</h2>
        <form onSubmit={this.handleSubmit}>
          <label>
            Name:
```

Assignment No: 6B

```
          <input
            type="text"
            name="name"
            value={this.state.name}
            onChange={this.handleInputChange}
            onKeyDown={this.handleKeyDown}
          />
        </label>

        <label>
          Email:
          <input
            type="email"
            name="email"
            value={this.state.email}
            onChange={this.handleInputChange}
            onKeyDown={this.handleKeyDown}
          />
        </label>

        <label>
          Gender:
          <select
            name="gender"
            value={this.state.gender}
            onChange={this.handleInputChange}
          >
            <option value="male">Male</option>
            <option value="female">Female</option>
            <option value="other">Other</option>
          </select>
        </label>

        <label>
          Message:
          <textarea
            name="message"
            value={this.state.message}
            onChange={this.handleInputChange}
            onKeyDown={this.handleKeyDown}
          />
        </label>

        <label>
          Subscribe to newsletter:
          <input
            type="checkbox"
            name="subscribe"
```

Assignment No: 6B

```
              checked={this.state.subscribe}
              onChange={this.handleInputChange}
            />
          </label>

          <div className="button-container">
            <button type="submit" onClick={this.handleSubmit}>
              Submit
            </button>
            <button type="button" onClick={this.handleReset}>
              Reset
            </button>
          </div>
        </form>
      </div>
    );
  }
}

export default RegistrationForm;
```

RegistrationForm.css

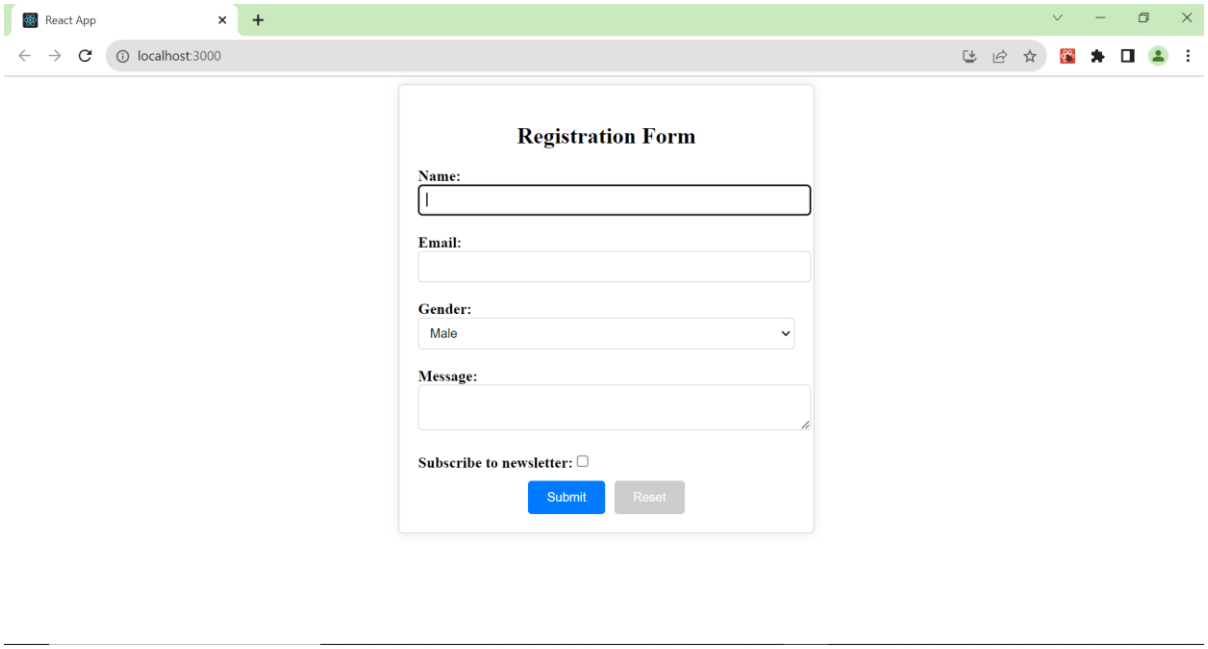
```
.registration-form {
    width: 400px;
    margin: 0 auto;
    padding: 20px;
    border: 1px solid #ccc;
    border-radius: 5px;
    background-color: #fff;
    box-shadow: 0px 0px 10px rgba(0, 0, 0, 0.1);
  }

  .registration-form h2 {
    text-align: center;
    margin-bottom: 20px;
  }

  .registration-form label {
    display: block;
    margin-bottom: 10px;
    font-weight: bold;
  }

  .registration-form input[type='text'],
  .registration-form input[type='email'],
  .registration-form select,
```

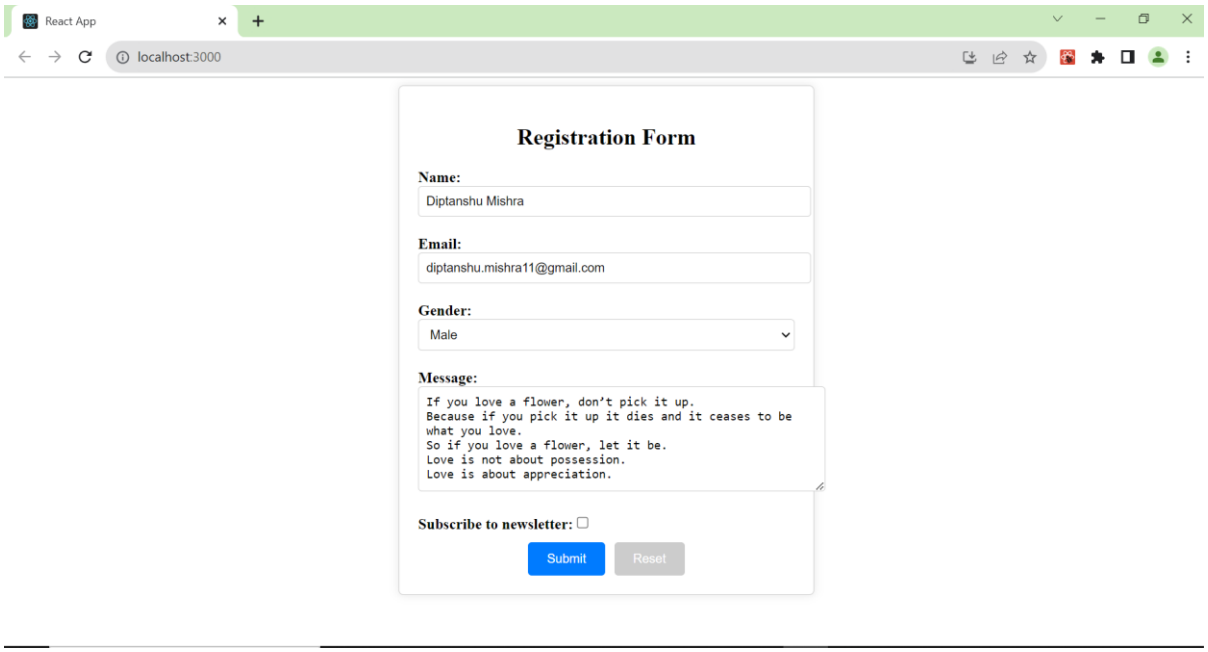
Assignment No: 6B

```css
.registration-form textarea {
  width: 100%;
  padding: 8px;
  margin-bottom: 10px;
  border: 1px solid #ccc;
  border-radius: 4px;
}

.registration-form input[type='checkbox'] {
  margin-right: 5px;
}

.button-container {
  text-align: center;
}

button {
  padding: 10px 20px;
  margin-right: 10px;
  background-color: #007bff;
  color: #fff;
  border: none;
  border-radius: 4px;
  cursor: pointer;
}

button:last-child {
  margin-right: 0;
  background-color: #ccc;
}

button:hover {
  background-color: #0056b3;
}
```

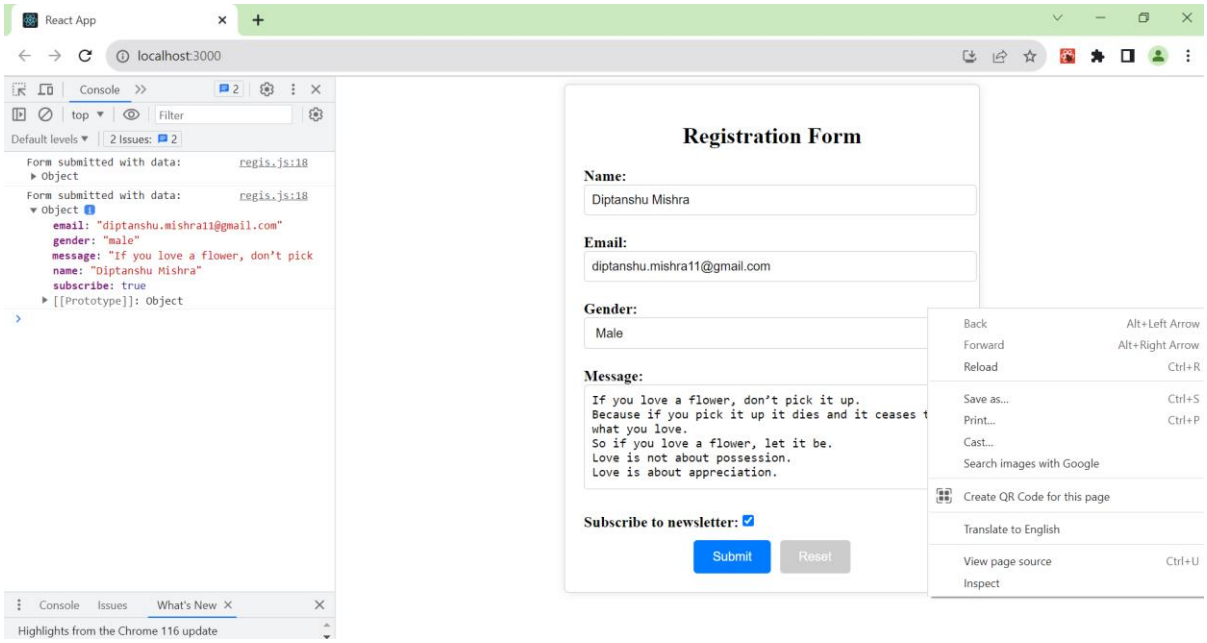
Output:

Assignment No: 6B





Assignment No: 6B



Conclusion: We studied that In React, an event is a user interaction or system-generated action that can trigger specific behavior in a React component and created a registration form which consisted of textbox, text area, selection input, check box, radio button, submit button and reset button handling onSubmit, onClick and keyDown events.

Assignment No: 7A

Aim: Write a JavaScript program to implement Router.

Theory:

In JavaScript, a router typically refers to a component or library used in web development for client-side routing. Client-side routing is a technique used in single-page applications (SPAs) to manage navigation and display different content without requiring a full page reload. Instead of requesting new HTML pages from the server, the router updates the view based on the URL changes, providing a smoother and more interactive user experience.

A JavaScript router is responsible for the following tasks:

1. Listening to URL Changes: It monitors changes to the browser's URL (usually the part after the '#' symbol, known as the fragment identifier or hash).

2. Mapping URLs to Components: It maps specific URLs or routes to corresponding JavaScript components or views that should be displayed when the URL matches a particular pattern.

3. Updating the View: When the URL changes and matches a defined route, the router updates the user interface by rendering the associated component or view.

4. Maintaining Browser History: It allows users to navigate forward and backward in the application's history using the browser's navigation buttons, even though the page itself doesn't fully reload.

Popular JavaScript libraries and frameworks like React Router, Vue Router, and Angular Router provide tools to implement client-side routing in web applications.

Code:

```
import React from 'react';

import { BrowserRouter as Router, Route, Link } from 'react-router-dom';
```

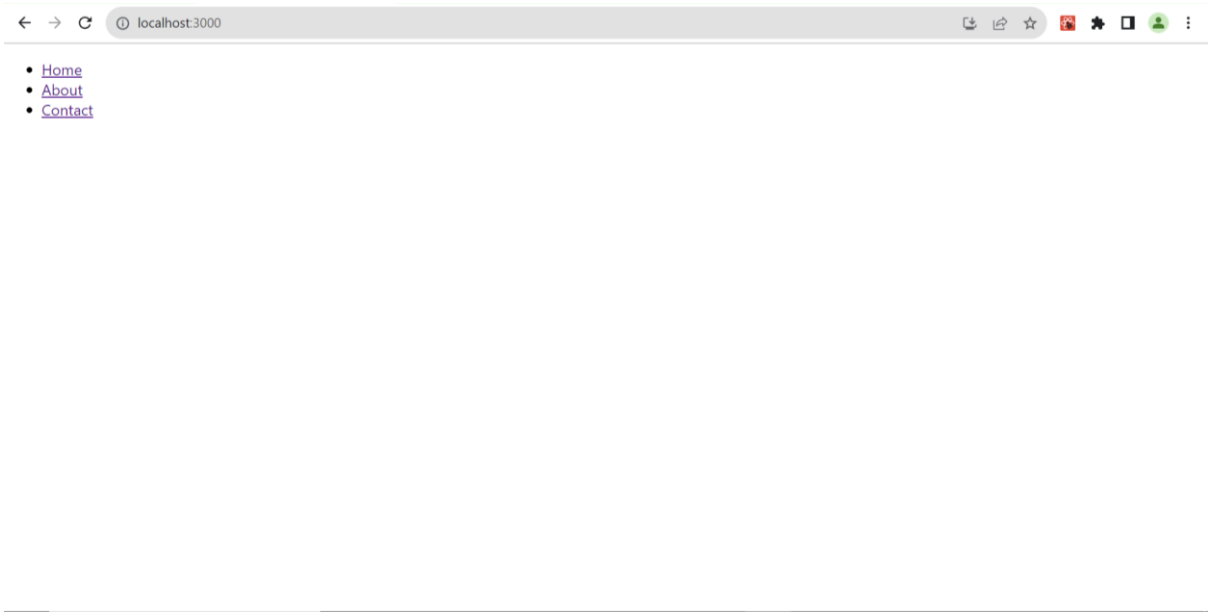
Assignment No: 7A

```
import { Switch } from 'react-router-dom'; // Import Switch separately

const Home = () => <div>Home Page</div>;
const About = () => <div>About Page</div>;
const Contact = () => <div>Contact Page</div>;


function App() {
  return (
    <Router>
      <nav>
        <ul>
          <li>
            <Link to="/">Home</Link>
          </li>
          <li>
            <Link to="/about">About</Link>
          </li>
          <li>
            <Link to="/contact">Contact</Link>
          </li>
        </ul>
      </nav>
      <Switch>
        <Route path="/" exact component={Home} />
        <Route path="/about" component={About} />
        <Route path="/contact" component={Contact} />
      </Switch>
```

Assignment No: 7A

```
    </Router>
  );
}

export default App;
```

Output:



Assignment No: 7A

- Home
- About
- Contact

Conclusion:

In this assignment we saw that a router typically refers to a component or library used in web development for client-side routing. Client-side routing is a technique used in single-page applications (SPAs) to manage navigation and display different content without requiring a full page reload.

Aim: Create a web application that displays a counter and allows the user to increment and decrement it using React hooks.

Theory:

In React.js, hooks are functions that allow you to “hook into” the state and lifecycle features of functional components. They enable functional components to manage state, side-effects, and other React features without needing to write a class component.

Instructions:

1. Set up a new React project using Create React App or your preferred method.

2. Create a functional component named `Counter` in a file named `Counter.js`.

3. Inside `Counter.js`, use the `useState` hook to manage a `count` state variable.

4. Render the `count` variable inside a `<div>` element.

5. Add two buttons, one to increment and one to decrement the `count` variable.

6. Implement the `onClick` handlers for both buttons using the `useState` hook to update the `count` variable accordingly.

7. Create an App component that includes the `Counter` component.

8. In the `App.js` file, import and render the `Counter` component.

9. Style your application using CSS if desired.

10. Test your application by running it using `npm start` or your preferred method.

Code:

```
Import React, { useState } from ‘react’;


Function Counter() {
 // Define a state variable ‘count’ and a function ‘setCount’ to update it
 Const [count, setCount] = useState(0);


 // Function to increment the count
 Const increment = () => {
  setCount(count + 1);
 };


 // Function to decrement the count (with validation to prevent negative values)
 Const decrement = () => {
  If (count > 0) {
   setCount(count – 1);
  }
 };


 Return (
  <div>
   <h2>Counter</h2>
   <p>Count: {count}</p>
   <button onClick={increment}>Increment</button>
   <button onClick={decrement}>Decrement</button>
  </div>
 );
```

}

Export default Counter;

Conclusion:

We studied that hooks are functions that allow you to “hook into” the state and lifecycle features of functional components. They enable functional components to manage state, side-effects, and other React features without needing to write a class component.

Assignment No-8

Aim:-

WAP to implement concept of REPL in command line.

Theory:-

The node:repl module exports the repl.REPLServer class. While running, instances of repl.REPLServer will accept individual lines of user input, evaluate those according to a user-defined evaluation function, then output the result. Input and output may be from stdin and stdout, respectively, or may be connected to any Node.js stream.

Instances of repl.REPLServer support automatic completion of inputs, completion preview, simplistic Emacs-style line editing, multi-line inputs, ZSH-like reverse-i-search, ZSH-like substring-based history search, ANSI-styled output, saving and restoring current REPL session state, error recovery, and customizable evaluation functions. Terminals that do not support ANSI styles and Emacs-style line editing automatically fall back to a limited feature set.

Commands and special keys
The following special commands are supported by all REPL instances:

.break: When in the process of inputting a multi-line expression, enter the .break command (or press Ctrl+C) to abort further input or processing of that expression.

.clear: Resets the REPL context to an empty object and clears any multi-line expression being input.

.exit: Close the I/O stream, causing the REPL to
exit. .help: Show this list of special commands.

.save: Save the current REPL session to a file: > .save ./file/to/save.js

.load: Load a file into the current REPL session. > .load
./file/to/load.js .editor: Enter editor mode (Ctrl+D to finish, Ctrl+C to
cancel).

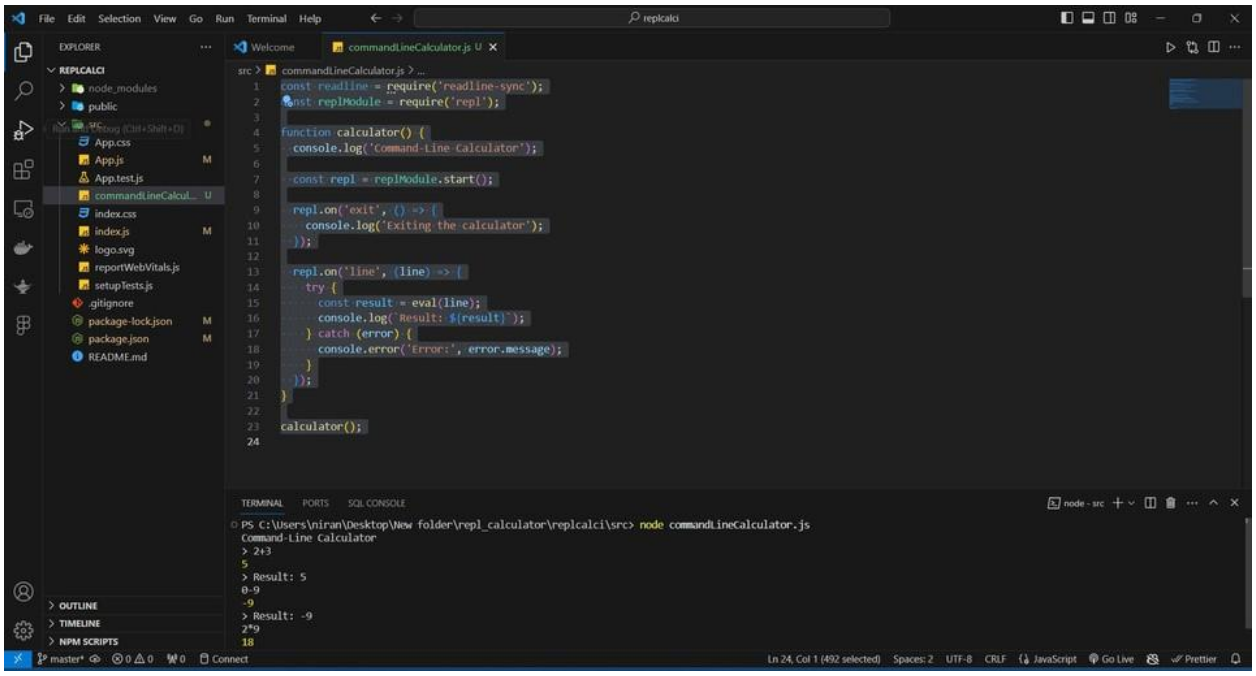
By default, all instances of repl.REPLServer use an evaluation function that evaluates JavaScript expressions and provides access to Node.js built-in modules. This default behavior can be overridden by passing in an alternative evaluation function when the repl.REPLServer instance is created.

Output:-

1)Code for Calculator.js

```
const readline = require('readline-sync');
const replModule = require('repl');

function calculator() {
  console.log('Command-Line Calculator');

const repl = replModule.start();

repl.on('exit', () => {
console.log('Exiting the
calculator');
});

repl.on('line', (line) => {
try{
const result = eval(line);
console.log(`Result: ${result}`);
} catch (error) {
console.error('Error:',
error.message);
}
}); }

calculator(

);
```

2)Output screenshots



Conclusion :-

We learnt that a “REPL” stands for “Read-Eval-Print Loop.” It’s an interactive programming environment that allows you to enter commands or code snippets, which are then read, evaluated, and the results printed back to you. This is commonly used in programming for quick experimentation and testing of code without having to write and run full programs. Many programming languages, like Python and JavaScript, have REPL environments that enable developers to interactively work with code.

Assignment : 9

Aim : WAP to implement Refs in React.

Theory :

Refs in React are a way to directly access and interact with DOM elements or React components. They provide a way to reference a specific object or object in your application, so that you can read values, trigger actions, or perform other tasks that would be difficult to achieve using React's standard dataflow and state scheduling methods role can help in situations such as access .
Use comments for React refs:
1. DOM Access:
Often Refs are used to interact directly with DOM elements. For example, you can access input values, set focus, or trigger animations.
2. Managing Third Party Libraries:
When integrating React with third-party libraries that require direct DOM manipulation, refs are needed. For example, if you use a charting library or a video player, you can use refs to interact with their API.

1. **Animating Elements:**

   You can use refs to trigger animations by changing CSS classes or properties. This is useful when you need precise control over animations.

2. **Scrolling and Measuring Elements:**

   You can use refs to scroll to specific elements on the page or measure their dimensions. This is helpful for building features like smooth scrolling or lazy-loading content.
3. **Integration with Non-React Code:**
   If you need to integrate React with non-React code, such as legacy JavaScript or libraries like D3.js, you can use refs to bridge the gap between React components and external code.
4. **Forms and Input Validation:**
   Refs can be useful for form handling and input validation. You can access input elements directly to check and manipulate their values or apply custom validation logic.

Creating Refs:

You can create a ref using the **React.createRef()** method (for class components) or the **createRef** hook (for functional components).

Accessing Refs:

You can access the DOM element or React component associated with a ref by using the **current** property of the ref object. This property holds the reference to the underlying element or component. You typically access the ref inside lifecycle methods
(for class components) or within the functional component itself.

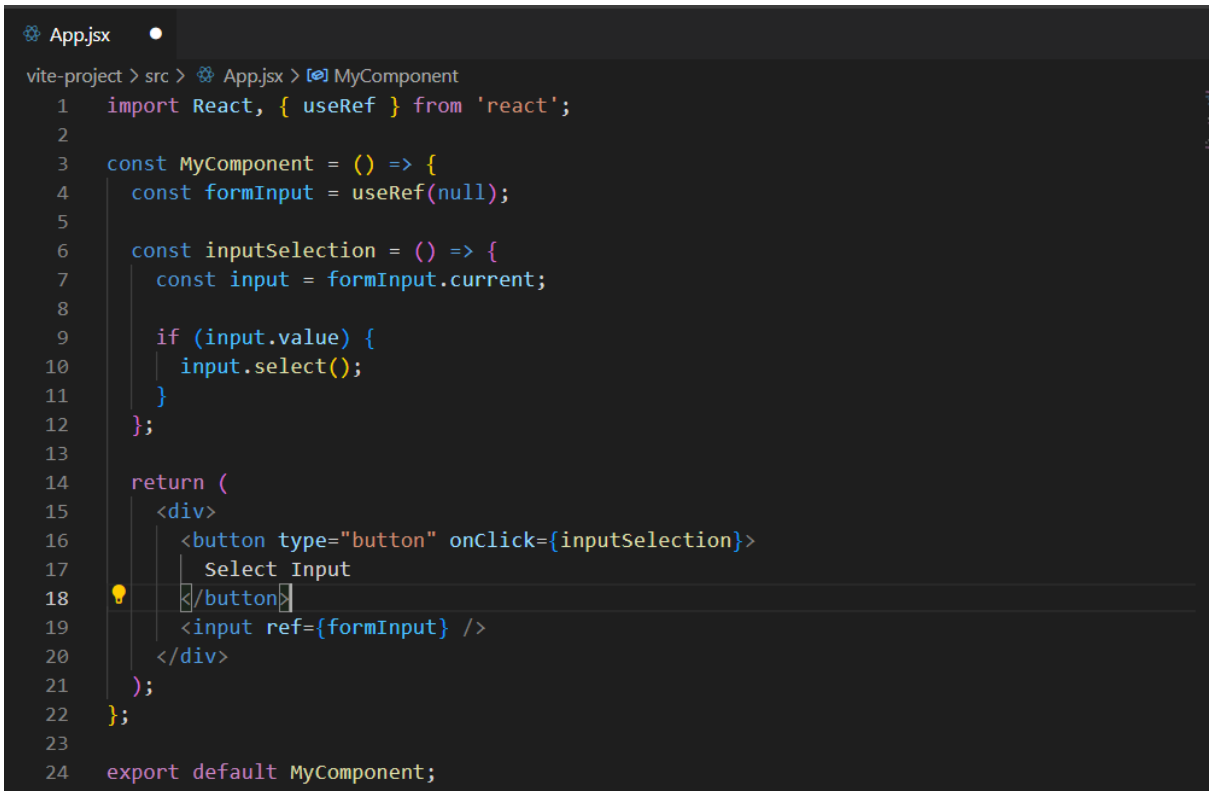
Forwarding Refs:

Sometimes, you may need to pass a ref from a parent component to a child component. This can be achieved using the "forwarding refs" technique. React provides the **forwardRef** function for this purpose. It allows you to pass a ref down the component hierarchy and access the child component's ref from a parent component.

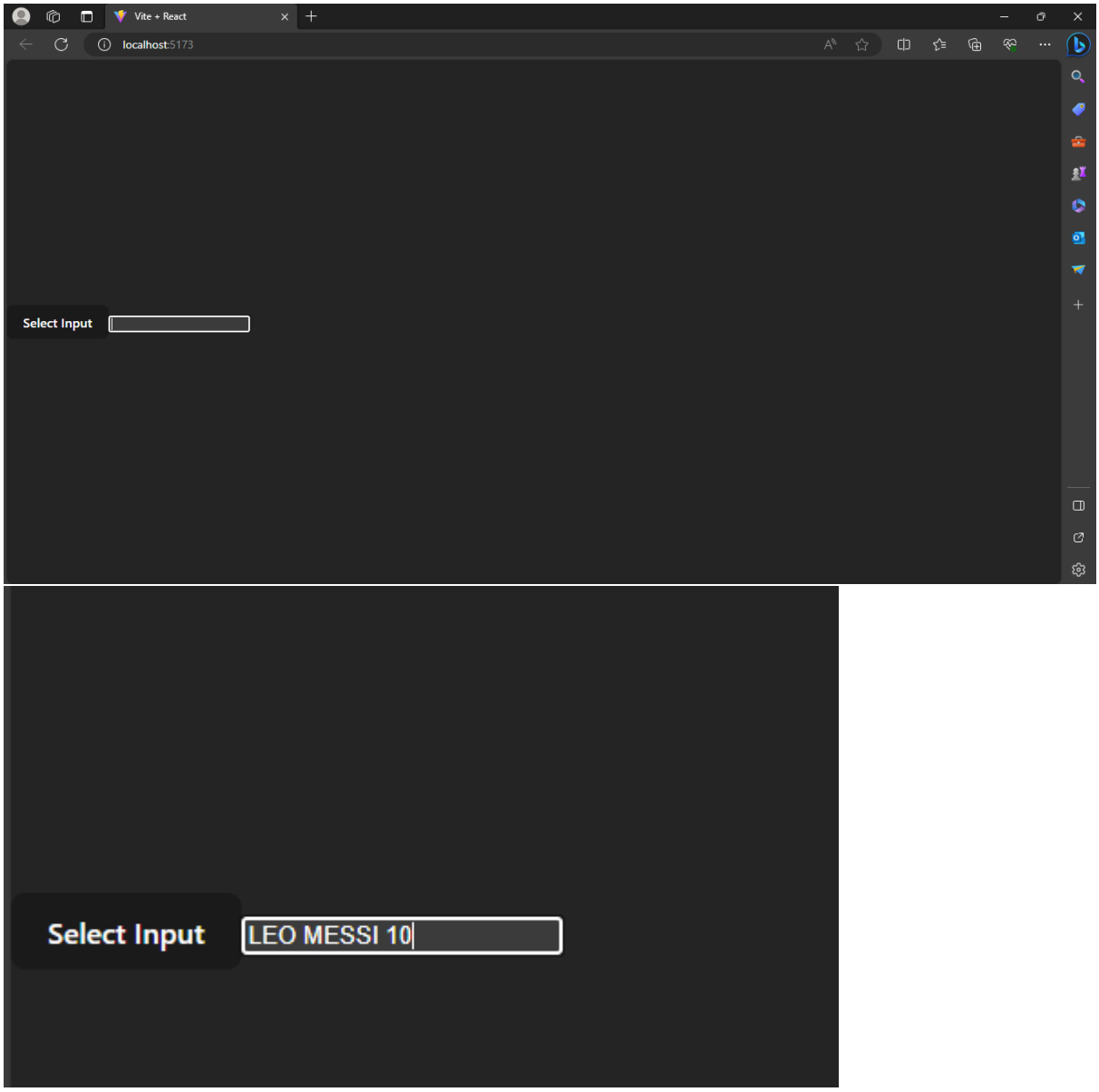
Callback Refs:

Callback refs are another way to work with refs, especially in cases where you need to perform additional operations when a ref is set. Instead of using the **createRef** or **useRef** functions, you define a callback function that will be called when the ref is attached to an element or component.

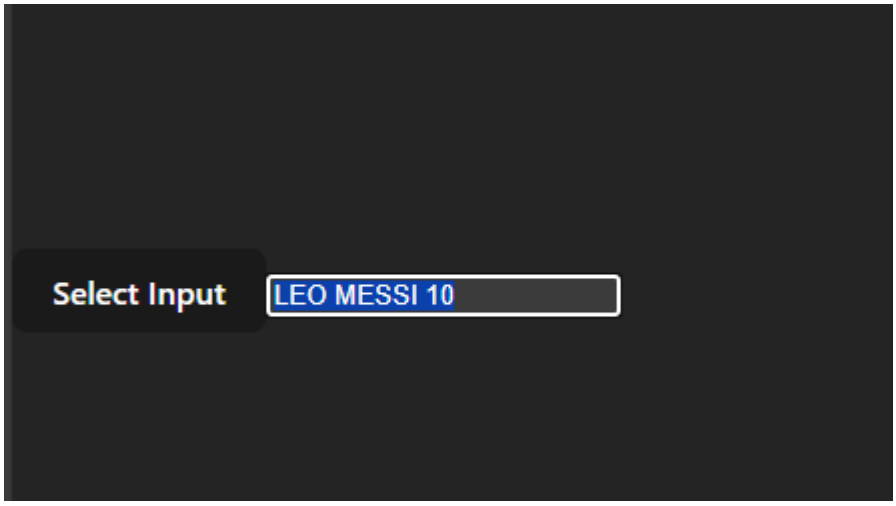
Code:



Output:



After clicking the button:



Conclusion :

We successfully demonstrated how React refs can be used to create, access, and manipulate DOM elements and components, showcasing their versatility within a React application.

Assignment 8

Aim: Write a program in Node JS to

a. Create a file

b. Read the data from file

c. Write the data to a file

d. Rename a file

e. Append the file

Theory:

File handling in Node.js involves performing various operations on files, such as creating, reading, modifying, renaming, appending, and deleting files. These operations are typically accomplished using the `fs` (file system) module, which is a built-in module in Node.js. Here's an explanation of each of these file handling operations in Node.js:

1. Creating a File:

- To create a new file, you can use the `fs.writeFileSync()` or `fs.writeFile()` function.

- `fs.writeFileSync(filename, data, [options])`: This method synchronously writes data to a file, creating the file if it doesn't exist.

- `fs.writeFile(filename, data, [options], callback)`: This method asynchronously writes data to a file, creating the file if it doesn't exist.

2. Reading a File:

- To read the content of a file, you can use the `fs.readFile()` function.

- `fs.readFile(filename, [options], callback)`: This method reads the content of a file asynchronously and provides the content in a callback function.

3. Modifying a File:

- To modify the content of a file, you can use functions like `fs.appendFile()` or `fs.writeFileSync()` if you want to overwrite the entire file.

- `fs.appendFile(filename, data, [options], callback)`: This method appends data to an existing file asynchronously.

4. Renaming a File:

- To rename a file, you can use the `fs.rename()` function.

Assignment 8

- `fs.rename(oldPath, newPath, callback)`: This method renames a file from `oldPath` to `newPath`.

5. Appending to a File:

- To append data to an existing file, you can use the `fs.appendFile()` function.

- `fs.appendFile(filename, data, [options], callback)`: This method appends data to an existing file asynchronously.

6. Deleting a File:

- To delete a file, you can use the `fs.unlink()` function.

- `fs.unlink(path, callback)`: This method deletes a file asynchronously.

This example demonstrates the complete life cycle of a file, including creation, reading, modification, renaming, and deletion.

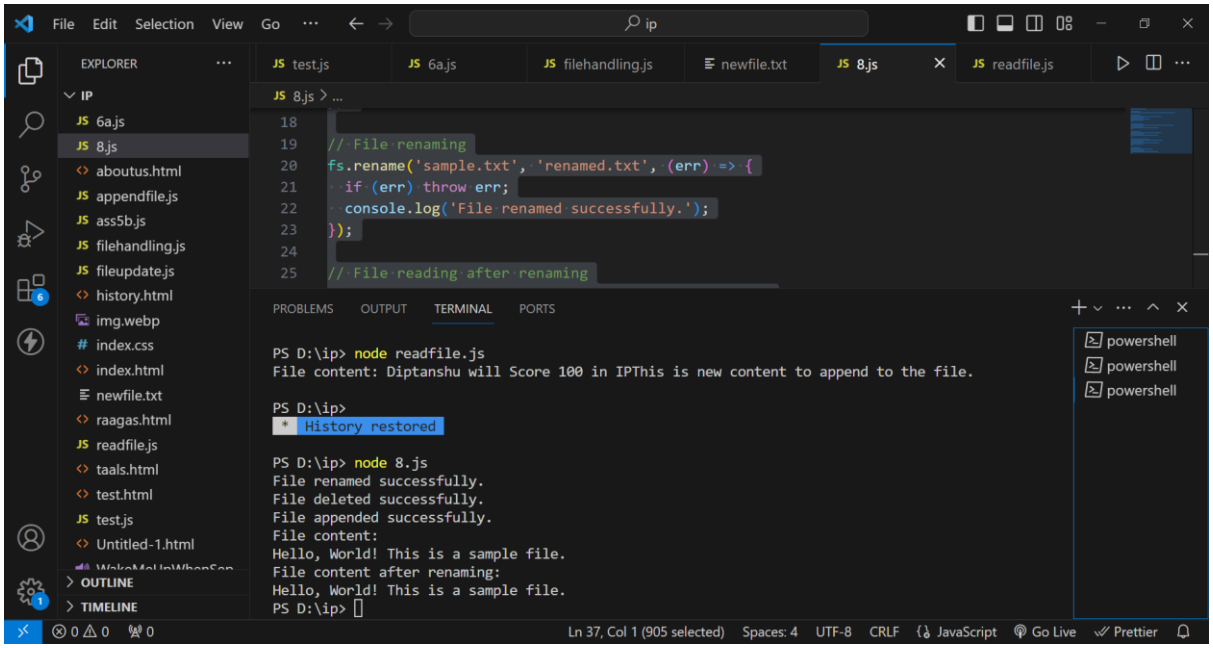
Code:

```
const fs = require('fs');

// File creation
fs.writeFileSync('sample.txt', 'Hello, World! This is a sample file.');

// File reading
fs.readFile('sample.txt', 'utf8', (err, data) => {
  if (err) throw err;
  console.log('File content:');
  console.log(data);
});

// File modification
fs.appendFile('sample.txt', '\nAppending some more text.', (err) => {
  if (err) throw err;
  console.log('File appended successfully.');
});

// File renaming
fs.rename('sample.txt', 'renamed.txt', (err) => {
  if (err) throw err;
  console.log('File renamed successfully.');
```

Assignment 8

```
});

// File reading after renaming
fs.readFile('renamed.txt', 'utf8', (err, data) => {
  if (err) throw err;
  console.log('File content after renaming:');
  console.log(data);
});

// File deletion
fs.unlink('renamed.txt', (err) => {
  if (err) throw err;
  console.log('File deleted successfully.');
});
```

Output:



Conclusion:

In this assignment we learnt that File handling in Node.js involves performing various operations on files, such as creating, reading, modifying, renaming, appending, and deleting files. These operations are typically accomplished using the `fs` (file system) module, which is a built-in module in Node.js.

Assignment No. 11

Aim: Write a web application that performs CRUD operations (database connectivity).

Theory:

Creating a web application that performs CRUD (Create, Read, Update, Delete) operations with database connectivity in the MERN stack (MongoDB, Express.js, React, and Node.js) involves various components and steps. Below, I'll provide a theoretical overview of how to design and implement such an application.

MERN Stack Overview:

1. MongoDB: MongoDB is a NoSQL database that will be used to store data related to the library, such as books, authors, and users.

2. Express.js: Express.js is a web application framework for Node.js that will handle routing and server-side logic.

3. React: React is a JavaScript library for building user interfaces. It will be used to create the front-end user interface for the library management system.

4. Node.js: Node.js is used as the server environment for the MERN stack application.

Core Functionalities:

1. Create (Add):

- Users should be able to add new books to the library database.
- Implement a user-friendly form for adding book details, including title, author, ISBN, and other relevant information.
- Validate and sanitize user input to ensure data integrity.
- Use POST requests to send the new book data to the server, which will then store it in the MongoDB database.

2. Read:

- Allow users to view a list of all books in the library.
- Implement a search feature to find books by title, author, or ISBN.
- Use GET requests to retrieve data from the MongoDB database and display it in the React front-end.

3. Update (Edit):

- Enable users to edit the details of existing books.
- Create an edit form that pre-fills with the existing book data.
- Use PUT requests to update book details in the database.

4. Delete:

- Provide an option to delete books from the library.
- Implement a confirmation dialog to prevent accidental deletions.
- Use DELETE requests to remove books from the database.

Architecture and Components:

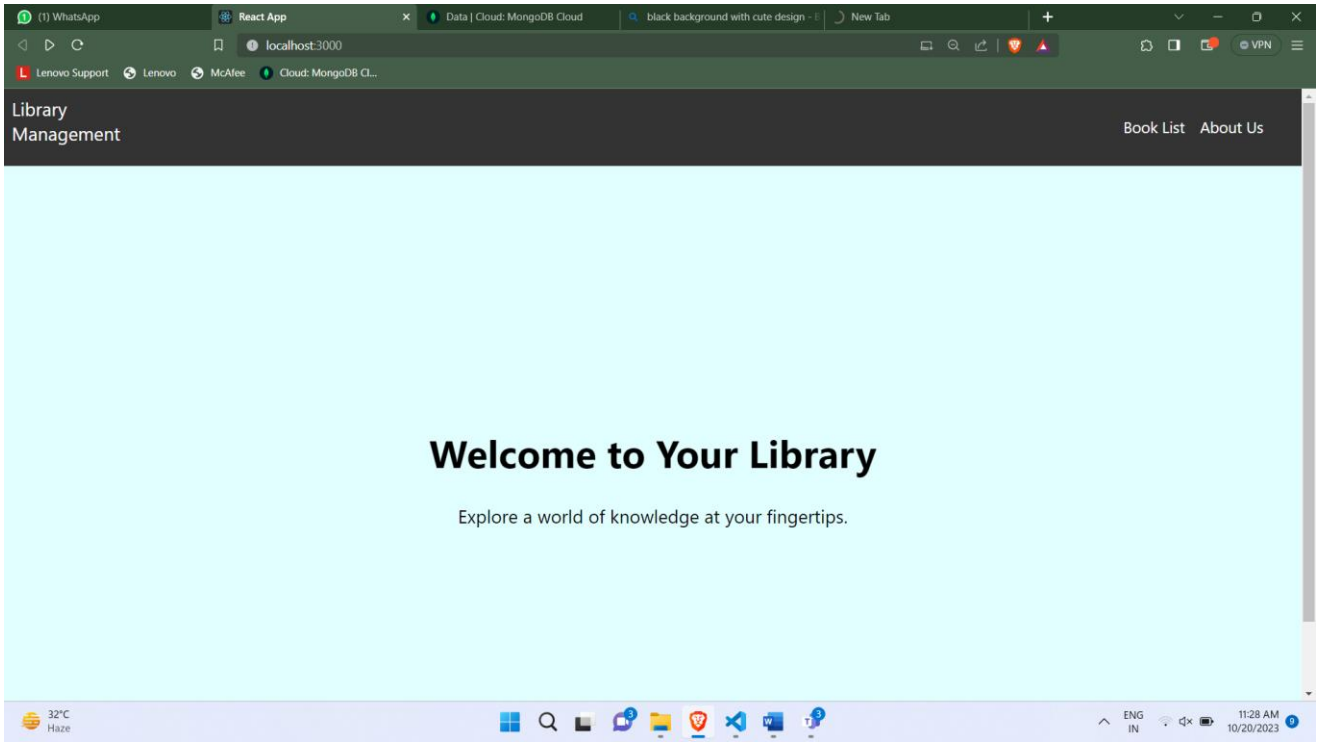
1. Server-side (Node.js and Express.js):
- Create an Express.js server that listens for incoming requests.
- Set up routes for handling CRUD operations (e.g., /api/books for managing books).
- Establish a connection to the MongoDB database using a Node.js MongoDB driver (such as Mongoose).

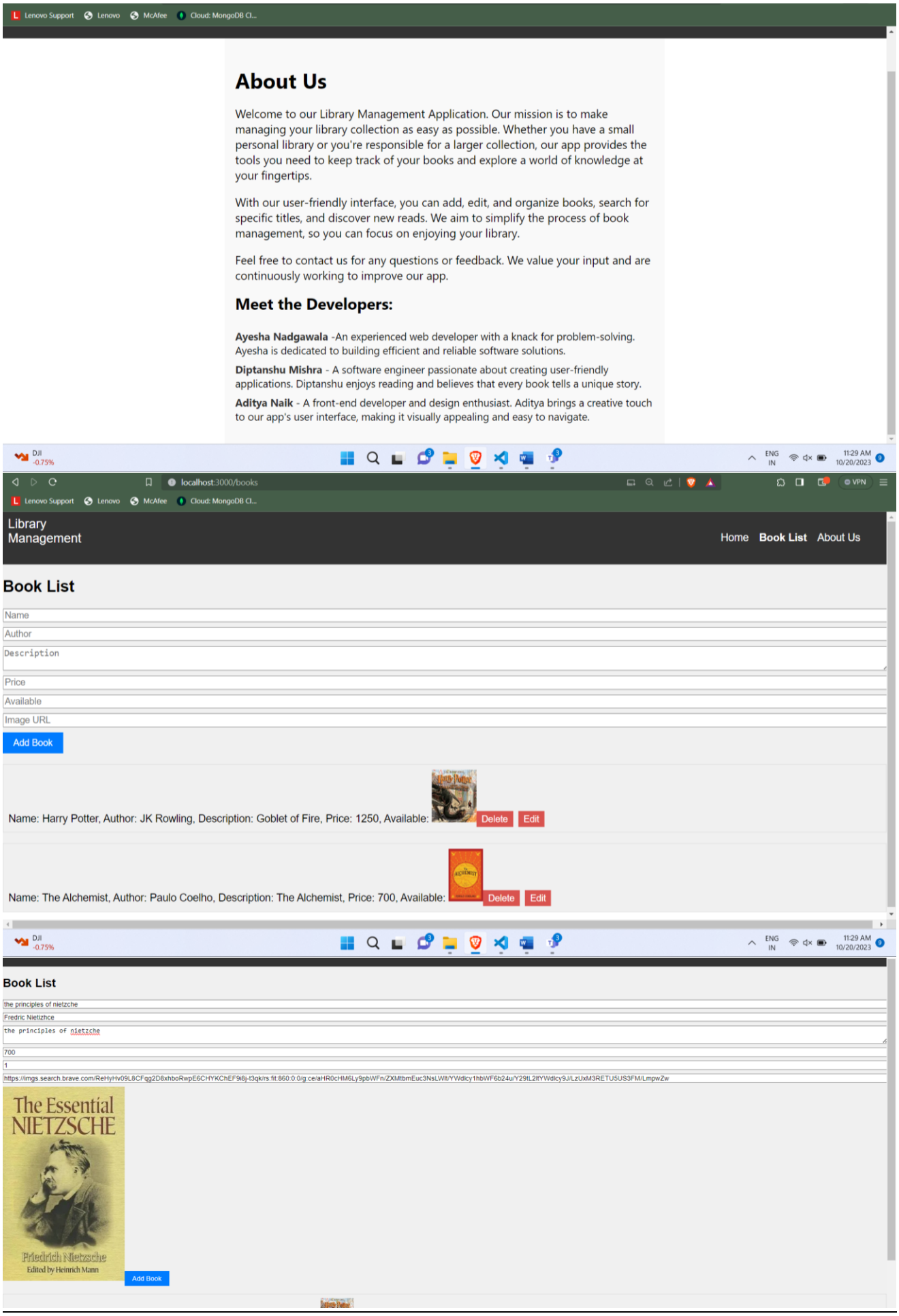
2. Database (MongoDB):
- Design a MongoDB schema for storing book information.
- Implement data models for books, authors, and users (if user authentication is required).
- Connect the Express.js server to MongoDB to perform CRUD operations on the database.

3. Front-end (React):
- Develop a user interface that allows users to interact with the library management system.
- Use components to create forms for adding, updating, and deleting books.
- Display a list of books with the option to view book details and edit or delete books.

4. API and Routing:
- Create RESTful API endpoints for handling CRUD operations (e.g., POST /api/books for adding, GET /api/books for reading, PUT /api/books/:id for updating, DELETE /api/books/:id for deleting).

Code:




Lenovo Support
Lenovo
McAfee
Cloud: MongoDB Cl...
About Us
Welcome to our Library Management Application. Our mission is to make managing your library collection as easy as possible. Whether you have a small personal library or you're responsible for a larger collection, our app provides the tools you need to keep track of your books and explore a world of knowledge at your fingertips.
With our user-friendly interface, you can add, edit, and organize books, search for specific titles, and discover new reads. We aim to simplify the process of book management, so you can focus on enjoying your library.
Feel free to contact us for any questions or feedback. We value your input and are continuously working to improve our app.
Meet the Developers:
Ayesha Nadgawala -An experienced web developer with a knack for problem-solving. Ayesha is dedicated to building efficient and reliable software solutions.
Diptanshu Mishra - A software engineer passionate about creating user-friendly applications. Diptanshu enjoys reading and believes that every book tells a unique story.
Aditya Naik - A front-end developer and design enthusiast. Aditya brings a creative touch to our app's user interface, making it visually appealing and easy to navigate.
DJI -0.75%
ENG IN
11:29 AM 10/20/2023
localhost:3000/books
VPN
Lenovo Support
Lenovo
McAfee
Cloud: MongoDB Cl...
Library Management
Home
Book List
About Us
Book List
Name
Author
Description
Price
Available
Image URL
Add Book
Name: Harry Potter, Author: JK Rowling, Description: Goblet of Fire, Price: 1250, Available:
Delete
Edit
Name: The Alchemist, Author: Paulo Coelho, Description: The Alchemist, Price: 700, Available:
Delete
Edit
DJI -0.75%
ENG IN
11:29 AM 10/20/2023
Book List
the principles of nietzche
Fredric Nietizhce
the principles of nietzche
700
1
https://imgs.search.brave.com/ReHyHv09L8CFqg2D8xhboRwpE6CHYKChEF9i8j-t3qk/rs:fit:860:0:0/g:ce/aHR0cHM6Ly9pbWFn/ZXMtbmEuc3NsLWt/YWdlcy1hbWF6b24u/Y29tL2ltYWdlcy9J/LzUxM3RETU5US3FM/LmpwZw
The Essential NIETZSCHE
Friedrich Nietzsche
Edited by Heinrich Mann
Add Book

Name: Harry Potter, Author: JK Rowling, Description: Goblet of Fire, Price: 1250, Available:  Delete Edit

Name: The Alchemist, Author: Paulo Coelho, Description: The Alchemist, Price: 700, Available:  Delete Edit

Name: the principles of nietzche, Author: Fredric Nietizhce, Description: the principles of nietzche, Price: 700, Available:  Delete Edit

---

Library Management

Home **Book List** About Us

Book List

Name

Author

Description

Price

Available

Image URL

Add Book

Name: Harry Potter, Author: JK Rowling, Description: Goblet of Fire, Price: 1250, Available:  Delete Edit

Name: The Alchemist, Author: Paulo Coelho, Description: The Alchemist, Price: 700, Available:  Delete Edit

Name: the principles of nietzche, Author: Fredric Nietizhce, Description: the principles of nietzche, Price: 700, Available:  Delete Edit

Edit Book

the principles of nietzche | Fredric Nietizhce | the principles of nietzche | 700 | | https://imgs.search.brave.c | Update

---

Library Management

Home **Book List** About Us

Book List

Name

Author

Description

Price

Available

Image URL

Add Book

Name: Harry Potter, Author: JK Rowling, Description: Goblet of Fire, Price: 1250, Available:  Delete Edit

Name: The Alchemist, Author: Paulo Coelho, Description: The Alchemist, Price: 700, Available:  Delete Edit

Name: the principles of nietzche, Author: Fredric Nietizhce, Description: edited description, Price: 700, Available:  Delete Edit

---

Conclusion:

In conclusion, developing a web application in the MERN stack to perform CRUD operations with database connectivity is a comprehensive process that involves designing a robust architecture, implementing core functionalities, ensuring security and data integrity, and deploying the application to make it accessible to users. By leveraging the strengths of MongoDB, Express.js, React, and Node.js, this framework offers a powerful and flexible foundation for creating modern, interactive web applications.

Written Assignment No: 1

Name: Diptanshu Mishra

Batch: T21

Roll no:78

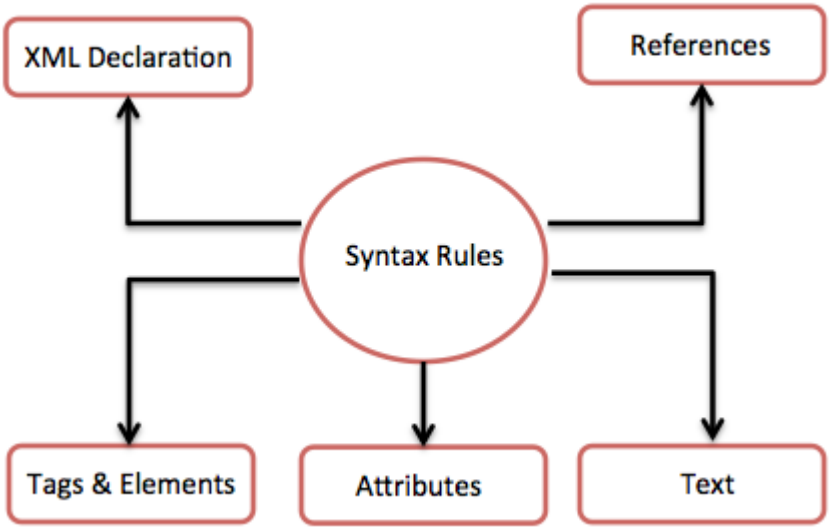
Q1. Compare XML and JSON.

Soln.

XML:

XML stands for Extensible Markup Language. XML tags identify the data and are used to store and organize the data, rather than specifying how to display it like HTML tags, which are used to display the data.

XML Syntax:



JSON:

JSON stands for JavaScript Object Notation JSON is a lightweight data-interchange format JSON is language independent. JSON is "self-describing" and easy to understand. JSON uses JavaScript syntax, but the JSON format is text only, just like XML. Text can be read and used as a data format by any programming language

Syntax:

The JSON syntax is a subset of the JavaScript syntax.

Data is in name/value pairs Data is separated by commas Curly braces hold objects ,square brackets hold arrays A name/value pair consists of a field name (in double quotes),

followed by a colon, followed by a value.

Example- “first name”: “Shreya”

JSON	XML
It is JavaScript Object Notation	It is Extensible markup language
It is based on JavaScript language.	It is derived from SGML.
It is a way of representing objects.	It is a markup language and uses tag structure to represent data items.
It does not provide any support for namespaces.	It supports namespaces.
It supports array.	It doesn’t support array.
Its files are very easy to read as compared to XML.	Its documents are comparatively difficult to read and interpret.
It doesn’t use end tag.	It has start and end tags.
It is less secured.	It is more secured than JSON.
It doesn’t support comments.	It supports comments.

Q2. Explain different types of arrow functions

Soln. Arrow functions are a concise way to write functions in JavaScript. They were introduced in ECMAScript 6 (ES6) and provide a more compact syntax compared to traditional function expressions. Arrow functions are especially useful for short, single-expression functions.

• These functions are also called as Arrow functions.

• There are 3 parts to a Lambda function −

• Parameters − A function may optionally have parameters.

• The fat arrow notation/lambda notation (=>): It is also called as the goes to operator.

• Statements − Represents the function's instruction set.

([param1, parma2,…param n])=>statement;

• Arrow functions remove the need to type out the keyword function every time you need to create a function.

• Instead, you first include the parameters inside the ( ) and then add an arrow => that points to the function body surrounded in { }.

Lambda Expression:

• It is an anonymous function expression that points to a single line of code.

• It is a compact alternative to a traditional function expression, but is limited and can't be used in all situations.

The different types of Arrow functions are:

1. Arrow function with no argument

If a function doesn't take any argument, then you should use empty parentheses.

```
let greet = () => console.log('Hello');
greet(); // Hello
```

2. Arrow function with one argument
   ```
   let greet = x => console.log(x);
   greet('Hello'); // Hello
   ```
   If a function has only one argument, you can omit the parentheses.

3. Arrow function as an expression

```
let age = 5;
let welcome = (age < 18) ?
() => console.log('Not Eligible') :
() => console.log('Eligible for Voting');
welcome();
```

4. Multiline Arrow functions

```
let sum = (a, b) => {
let result = a + b;
return result;
}
let result1 = sum(5,7);
console.log(result1); // 12
```

Advantages of Arrow Functions

- Arrow functions reduce the size of the code.
- The return statement and function brackets are optional for single-line functions.

- It increases the readability of the code.
- Arrow functions provide a lexical this binding. It means, they inherit the value of “this” from the enclosing scope. This feature can be advantageous when dealing with event listeners or callback functions where the value of “this” can be uncertain.

Limitations of Arrow Functions

- Arrow functions do not have the prototype property.
- Arrow functions cannot be used with the new keyword.
- Arrow functions cannot be used as constructors.
- These functions are anonymous and it is hard to debug the code.
- Arrow functions cannot be used as generator functions that use the yield keyword to return multiple values over time.

Q3. What is DNS? Explain working of DNS.

Soln.

DNS:

The domain name system (DNS) is a naming database in which internet domain names are located and translated into Internet Protocol (IP) addresses. The domain name system maps the name people use to locate a website to the IP address that a computer uses to locate that website.

For example, if someone types "example.com" into a web browser, a server behind the scenes maps that name to the corresponding IP address. An IP address is similar in structure to 203.0.113.72.

Web browsing and most other internet activities rely on DNS to quickly provide the information necessary to connect users to remote hosts. DNS mapping is distributed throughout the internet in a hierarchy of authority. Access providers and enterprises, as well as governments, universities and other

organizations, typically have their own assigned ranges of IP addresses and an assigned domain name. They also typically run DNS servers to manage the mapping of those names to those addresses. Most Uniform Resource Locators (URLs) are built around the domain name of the web server that takes client requests.

How DNS works:

DNS servers convert URLs and domain names into IP addresses that computers can understand and use. They translate what a user types into a browser into something the machine can use to find a webpage. This process of translation and lookup is called *DNS resolution*.

The basic process of a DNS resolution follows these steps:

1. The user enters a web address or domain name into a browser.
2. The browser sends a message, called a *recursive DNS query*, to the network to find out which IP or network address the domain corresponds to.
3. The query goes to a recursive DNS server, which is also called a *recursive resolver*, and is usually managed by the internet service provider (ISP). If the recursive resolver has the address, it will return the address to the user, and the webpage will load.
4. If the recursive DNS server does not have an answer, it will query a series of other servers in the following order: DNS root name servers, top-level domain (TLD) name servers and authoritative name servers.
5. The three server types work together and continue redirecting until they retrieve a DNS record that contains the queried IP address. It sends this information to the recursive DNS server, and the webpage the user is looking for loads. DNS root name servers and TLD servers primarily redirect queries and rarely provide the resolution themselves.
6. The recursive server stores, or caches, the A record for the domain name, which contains the IP address. The next time it receives a request for that domain name, it can respond directly to the user instead of querying other servers.
7. If the query reaches the authoritative server and it cannot find the information, it returns an error message.

The entire process querying the various servers takes a fraction of a second and is usually imperceptible to the user.

DNS servers answer questions from both inside and outside their own domains. When a server receives a request from outside the domain for information about a name or address inside the domain, it provides the authoritative answer.

When a server gets a request from within its domain for a name or address outside that domain, it forwards the request to another server, usually one managed by its ISP.

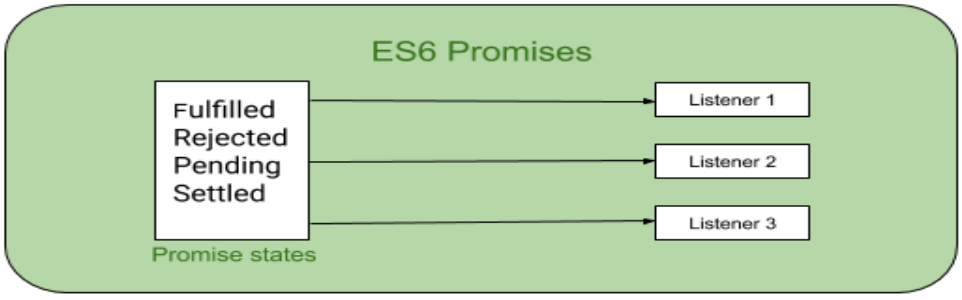
Q4. Explain Promises in ES6.

Soln.

Promises are a way to implement asynchronous programming in JavaScript (ES6 which is also known as ECMAScript-6). A Promise acts as a container for future values. Like if you order any food from any site to deliver it to your place that order record will be the promise and the food will be the value of that promise.

So, the order details are the container of the food you ordered. Let's explain it with another example. You order an awesome camera online. After your order is placed you receive a receipt of the order. That receipt is a Promise that your order will be delivered to you. The receipt is a placeholder for the future value namely the camera. Promises used in JavaScript for asynchronous programming.

For asynchronous programming, JavaScript uses callbacks, but there is a problem using the callback which is callback hell (multiple or dependent callbacks) or Pyramid of Doom. Using the ES6 Promise will simply avoid all the problems associated with the callback.



Need of Promises: The Callbacks are great when dealing with basic cases. But while developing a web application where you have a lot of code. Callbacks can be great trouble. In complex cases, every callback adds a level of nesting which can make your code really messy and hard to understand. In simple words, having multiple callbacks in the code increases the complexity of the code in terms of readability, executability, and many other terms. This excessive nesting of callbacks is often termed Callback Hell.

Example: Callback Hell.

To deal with this problem, we use Promises instead of callbacks.

Making Promises: A Promise is created when we are unsure of whether or not the assigned task will be completed. The Promise object represents the eventual completion (or failure) of an async(asynchronous) operation and its resulting value. As the name suggests from real life itself, a Promise is either kept or broken. A Promise is always in one of the following states:

- **fulfilled:** Action related to the promise succeeded.
- **rejected:** Action related to the promise failed.
- **pending:** Promise is still pending i.e not fulfilled or rejected yet.
- **settled:** Promise has been fulfilled or rejected

Assignment No: 2

Refs (short for references) are a feature in React, a popular JavaScript library for building user interfaces. Refs provide a way to access and interact with the real DOM elements or React components directly. They are typically used when you need to perform actions such as reading values from an input field, focusing on an input field, or animating elements imperatively.

Here's a short note on when to use refs and when not to use refs:

When to use Refs:

1. Managing focus: Use refs to manage the focus of input fields or other elements. For example, you can use refs to autofocus an input element when a component mounts.

2. Accessing DOM properties: When you need to access properties or methods of a DOM element that React's synthetic event system doesn't expose, such as measuring an element's dimensions or triggering animations.

3. Integrating with third-party libraries: When integrating React with non-React code or third-party libraries that expect direct access to DOM elements, using refs can be helpful.

4. Managing text selections: Refs can be used to select text within an input field or textarea programmatically.

When not to use Refs:

1. State and Props: In React, it's generally better to manage your application's state and props to control the rendering of components. Refs should be used sparingly when you can't achieve a task through state and props.

2. Avoid direct DOM manipulation: Refs should not be used to manipulate the DOM directly whenever possible. React is designed to abstract away direct DOM manipulation to maintain a declarative and predictable programming model.

3. Avoid overusing refs: Overusing refs can make your code less declarative and harder to understand. Try to use refs only when necessary for specific use cases.

Assignment No: 2

NPM (Node Package Manager) is a package manager for JavaScript and Node.js. It allows developers to easily install, manage, and share JavaScript libraries and tools. NPM is typically used to:

1. Install Packages: Developers use NPM to install packages and dependencies required for their projects. These packages can be libraries, frameworks, or tools that extend the functionality of JavaScript applications.

2. Manage Dependencies: NPM helps manage project dependencies by maintaining a `package.json` file. This file lists all the dependencies required for a project, including their versions.

3. Version Control: NPM provides version control for packages, allowing developers to specify the exact version or version range of a package to ensure consistent behavior across different environments.

4. Scripts: NPM allows developers to define and run custom scripts as part of the project's build, test, or deployment processes.

5. Publish Packages: Developers can publish their own JavaScript packages to the NPM registry, making them available for others to use.

NPM is a crucial tool for JavaScript development, and it simplifies the management of project dependencies and the distribution of JavaScript code.

REPL stands for Read-Eval-Print Loop. It's an interactive programming environment that allows developers to enter code, have it executed, and see the results immediately. Node.js provides a built-in REPL environment, which is a powerful tool for testing and experimenting with JavaScript code without the need for writing full programs.

In a REPL, you can:

- Read: Enter JavaScript code line by line or in blocks.

Assignment No: 2

- Eval: Have the entered code evaluated and executed.

- Print: See the output or result of the code execution.

- Loop: Repeat the process by entering more code as needed.

The Node.js REPL is especially useful for debugging and quick code prototyping. It's a handy way to test small code snippets or explore the behavior of JavaScript functions and expressions interactively.

To start a Node.js REPL, you can simply open your terminal and type `node`. This will launch the REPL environment, allowing you to enter JavaScript code and see the results in real-time.

Routing in Express.js:

Routing in Express.js is the process of defining how the application responds to client requests for specific URLs (or routes). It allows you to map different HTTP request methods (such as GET, POST, PUT, DELETE) to specific functions (route handlers) that handle those requests.

Here's an explanation along with an example:

```
const express = require('express');
const app = express();
const port = 3000;

app.get('/', (req, res) => {
   res.send('Hello, World!');
});
```

Assignment No: 2

```
app.get('/about', (req, res) => {
  res.send('This is the About page.');
});

app.get('/contact', (req, res) => {
  res.send('Contact us at contact@example.com.');
});

app.listen(port, () => {
  console.log(`Server is running on port ${port}`);
});
```

In the example above:

1. We import the Express.js framework and create an Express application.

2. We define routes using the `app.get` method. In this case, we have routes for the root URL (`/`), the About page (`/about`), and the Contact page (`/contact`).

3. Each route has a corresponding route handler function that gets executed when a request matches that route. The handler sends a response to the client.

4. Finally, we start the Express server on port 3000, and it listens for incoming HTTP requests.

When a client makes a request to one of the defined routes, Express.js matches the URL to the appropriate route handler, and that handler sends a response back to the

client. This is the fundamental concept of routing in Express.js, and it allows you to build web applications with different routes and behavior for each route.

Conclusion: In this assignment we learnt about Refs, NPM, REPL and Routing in ExpressJs.